UiO : Department of Informatics
University of Oslo

The Hydra

An automated disaster recovery solution for the cloud

Ke Huang
Network and System Administration
master thesis spring 2013

The Hydra

Ke Huang
Network and System Administration

May 23, 2013

Abstract

Many businesses utilize disaster recovery plan to ensure service continuity. However, current disaster recovery mechanisms generally require manual intervention to restore the failed system and cannot prevent from service interrupt since it may take minutes to bring backup system online after a failure.

To aid in this situation, this thesis proposed a disaster recovery solution which aims to achieve an automated, adaptable and reliable recovery process on a cloud based system. First, a novel mechanism has been proposed to take advantage of cloud service and recover automatically from unforeseeable system failure, without complicated manually management. It enhances the reliability of services by providing a distributed self-organizing structure which can detect and heal failures spontaneously. Then, based on the mechanism a model has been designed and thoroughly described. Finally, based on the model, a prototype was implemented to demonstrate the performance benefits from utilizing this mechanism in cloud based disaster recovery.

Acknowledgements

My gratitude to Kyrre M. Begnum, my supervisor, who initially inspired me with this interesting idea and provided valuable guidance, inspiration and encouragement. His continuous guidance and support helped me during the time of research and writing of this thesis. I feel great to have him be my supervisor and I always appreciate for his outstanding support.

A special thanks to Aleen Frisch. Her extraordinary talent, unlimited patience and continuous encouragement motivated me throughout the two years study.

My sincere thanks to Hårek Haugerud and all the lecturers, professors for their guidance and their excellent efforts to overcome the challenges I faced throughout whole period of this master program. Two years study has certainly been a wonderful learning experience for me.

Oslo, May 2013
Ke Huang

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the massive growth of cloud-based services, there are even more options of storing and sharing information on the Internet. Components including servers and network management solutions have been delivered as services by many cloud service providers. With the obvious benefits in flexibility, high availability and pay-as-you-use basis model, cloud infrastructures have been purchased by many companies as virtual resources to run their sites rather than to host them on their own servers. Over the last years, a large number of cloud service providers have been developed, such as Amazon EC2, Rackspace, and Google Compute Engine etc. Nowadays, thousands of customers of all sizes, ranging from individual users to large companies, are depending on cloud services to host their sites and store their business data.

Along with the accelerating development of cloud services and technologies, some critical issues have been raised, such as the unpredictable connectivity issues and unexpected outages. Even the most popular hosting providers, such as Amazon Web Services and Rackspace, still occasionally suffer from service interruptions or failures [1].

On 22 May 2013, a flood in parts of Eastern Norway has hit the media group Amedia's data center at Kjeller outside Lillestrøm. The IT system media group and its around 50 local newspapers are impacted by this outrage [2].

On 7 Jan. 2013, Rackspace hosting suffered a widespread of email and application outages[3]. On Christmas Eve 2012. The cloud computing outrage of Amazon Web Services took down many services, especially Netflix, for more than 23 hours [4].

There are even more reports of various downtimes of the cloud services caused by power outage, (DDoS) attacks, machine malfunction, human error etc. [5, 6, 7, 8, 9, 10]. In addition to having caused services shutting down, some crash disasters even have destroyed many customers' data permanently [11].

System failures like these are bound to happen. The cloud services are already familiar to us, but failure recovery for cloud based services is relatively new. Although, in theory, the cloud model is more failure-tolerant by employing hardware redundancy and/or distributing applications in different data centers around the world, it's still far from total immunity of failures.

To help the customers of cloud services to deal with such failures better, many cloud service providers, such as Amazon's cloud, have started to introduce new features that would deal with and minimize the impacts of disasters. Details of these features will be covered later in the background chapter 2.1.5.

However, even with these new failure-handling features, to build up an operational failure-handling system on top of them still presents a great challenge to many cloud service customers, for these features, although being nonetheless ample and powerful, have following drawbacks:

1. They often expose too much technical details. Therefore, to use them properly requires strong cloud related technical background and considerable amount of engineering effort. For example, many of these failure-handling features expect the managed services to be distributed over multiple regions. Such technical detail could be easily overlooked if the companies which use cloud service, especially small companies, lack of deep understanding of how cloud works. Many companies, although using cloud service, never distributed new services beyond their first AWS data center [12]. Consequently, these distribution based failure-handling features would never work. On the other hand, improper usage of these features, which is prone to happen, could cause uncontrollable complexity, confusion, and errors, which, in turn, lead to more unplanned downtime.

2. They are provided not as a systematic solution but a collection of features. System administrator then need to develop tools, often single-purposed ones, based on them. Being limited by the single or a few features in the collection of which they are built on top, these single-purposed tools each usually could only deal with but one specific situation, not to mention their poor ability of handling works crossing operating-systems and infrastructures of datacenters.

3. The current single-purposed tool based solution, as mentioned in above, which is "De facto" mainstream approach that is made on top base. These new failure-handling features, is still vulnerable to the failures occurred simultaneously to both on-line and standby systems, for it still depends on redundant resources in predefined locations. And even more, it costs extra since the customers need to pay for hot backups which stay idle at most of the time in cloud [13]. Also, it does not provide a means to spontaneously handle failures, which means, the costumers have to maintain a team of engineers at standby to handle potential failures, or to hire a third party to do so.

To aid in this situation, administrators need the comprehensive solution to help them automating the whole process of failure handling of cloud services. This paper, while providing significant improvements, explores a novel mechanism to take advantage of cloud service and recover automatically from unforeseeable system failure, without complicated manually management. The design is to enhance the reliability of services by providing a distributed self-organizing structure which can detect and heal failures spontaneously. Moreover, it provides a degree of portability and usability for its built based on a set of wide-used administration tools such as puppet, which is user friendly and capable of running on various operating systems by itself.

1.1 Problem Statement

The following problem statement guides the direction of this thesis.

How can we achieve a mechanism to perform an automated, adaptable and reliable site disaster recovery based on cloud services?

The *automated* as used in the problem statement covers the features include auto-growth, auto-detecting and self-healing. The site has been designed to be auto-growth like a cell. Every single site has the ability to replicate itself and create the entire system without external intervention. Also, with unattended monitoring procedure, anomalies are automatically detected and confirmed. Once the site failure has been confirmed, restore process will then be triggered. In the mean time, the error signal is sent and propagated to all zones. Unavailable site will then be terminated while its replica has being generated and finds an appropriate region to live.

Adaptable refers to use the cloud features of creating the new sites on an as-needed

basis. Instead of having backup sites sitting idly in cloud all the time, new sites will only be created and brought online once the outrage of old site has been confirmed.

Reliable represents the features of *stability* and *data availability*.

Stability comes from the multi-site solution. There are multiple availability zones in each region of the cloud. The sites will be delivered to multiple availability zones and different regions. It is important that the sites work for the same application are geographically distributed, which ensuring that a disaster occurred in one geographic regions will not cause service disruption of the whole system. However, this geographic distribution may increase network latency. In order to avoid such latency, the second policy here is to always establish new cloud regions in a minimal geographic distance to the users that the cloud regions serve to.

The *data availability* is also ensured by the multi-site solution. When a new site created in an availability zone, it automatically synchronizes and replicates the most recently data of whole system, hence, the data is always distributed and unlikely to become available because one individual site fails.

Chapter 2

Background

2.1 Autonomic Computing

Computing systems are becoming increasingly complicated. With the growing complexity of managing IT infrastructures, the interconnectivity and integration in the configuration of different software, it is getting more and more difficult to keep the systems configuring and running only by human efforts.

In order to cope with this problem, IBM has introduced the concept of autonomic computing in October 2001[14]. In the manifesto released by IBM, it points out, the main challenge of future IT industry is a software complexity crisis. The difficulty of system development, configuration and management are getting beyond the administration of single software environments. The computing system today is becoming too massive and complicated to manage. And the complexity, confusion, and system errors that result from improper configuration and management can contribute to more unplanned system problems. This situation imposed additional management workload associated with the administration is approaching the limitation of human capability [15].

The only option remaining is autonomic computing. An autonomic computing system, also known as self-management system, is a computer system that can manage itself according to the objectives of the administrator. It runs itself, detects and adjusts to varying circumstances automatically without the assistance of manual interventions.

Research in autonomic computing is still in early stage, it overwhelms the capabilities of existing tools and methodologies. Over 20 workshops and conferences contributed to this research over the last two years [16], autonomic computing pro-

vides a wide domain for researchers in many areas of computer science, including software architecture and artificial intelligence.

The essence of autonomic computing system is to create a system which has a life-like properties. Components integrate themselves in the system just like the procedure of cells establish themselves in the human body. They have properties of self-configuration, self-optimization, self-healing and self-protection[17, 18].

2.1.1 Self-Management

The aim of self-management of autonomic computing systems is to relieve the administrators from the details of system configurations, and let the system continuously adjusts and runs at its highest performance by itself. To be autonomic, a system first needs to know itself, know its environment, that means all components included in the system must possess a system identity [16]. The autonomic system need continuously to monitor its own behavior, and check for the changes of its conditions and environment. When it detects errors, the system will recover to previous version and try to isolate the errors by its problem determination algorithm. [17].

As mentioned above, there are four aspects of self-management along with autonomic computing. The four aspects, which frequently cited by IBM's manifesto, are self-configuration, self-optimization, self-healing and self-protection. Early autonomic systems usually consider these aspects separately. The traditional method is to address problems which related to different aspects by specific solutions, however, with the automation technologies' improvement, these aspects will be eventually combined and integrated to a general architecture.

The explanation of the four aspects is as follows:

Self-Configuration

An autonomic computing system must configure and reconfigure itself to adapt varying circumstances. The configuration of autonomic system follows high-level policies which consider what is desired, instead of how it will be accomplished. When a new component integrates into a system, it will automatically adapt to the overall configuration of the system and then register itself to the system. By this mechanism, other components can learn about the new one, get the knowledge to use it or changed themselves to work with it [19].

Self-Optimization

At the present, the complexity of middleware is increasing rapidly, a database system which may have plenty of tunable parameters. Consequently, even the most skilled IT administrators may find it impossible to configure them properly and keep the entire system in the optimal performance.

Instead of staying for status quo, an autonomic system can continually seek ways to monitor and optimize its performance or cost. This mechanical based on the biological concept in which autonomic computing is the simulation of human body. Much like the brain modifies its circuitry to functional better during daily practice and learning. Autonomic systems will learn to tune their own parameters properly and try to upgrade their functions to meet end user's needs with none or minimal human interference.

Self-Healing

Since autonomic computing system normally works in dynamic environments that mostly changes during routine operations. It is vital to continually assess system status, and moreover, to discover malfunctions, diagnose and react to system disruption, and to apply appropriate corrections automatically[20, 16]. Components will continually monitor themselves to ensure the system integrity. Using problem-diagnosis component, the health of individual component is processed into assessment, the failed component is determined and isolated. After the first stage of monitoring and analysis, system is able to recover from failures by fixing, replacing, rebooting, or isolating the faulty component.

The objective of self-healing system is to minimize all impacts of system outages and introduce the fixed component back to the system without disruption of the service. The system may need to predict the problems and hence take steps to prevent failures beforehand.

Self-Protection

It is not impossible for an autonomic computing system to exit in an isolate environment without any outside influence. For a system to be self-protecting, try to avoid failures caused by events such as malicious attacks, unauthorized access and misoperation from users, the system must have the ability to defend itself. It needs to continually monitoring all parts of the system according to security po-

lices, and take appropriate actions to keep the compliance of components. A recent security solutions offering by IBM is to provide effectual rules of automating security compliance management [21, 22]. Secondly, the system need to handle threats whenever they occur, protect itself from the attacks from anywhere [20] and report such activities.

Autonomic computing system that capable of self management has many benefits, as well as to address complexity of modern computing system, it reduces system maintenance burden and lowers cost of administration. But the challenge of exploring various aspects of autonomic computing and achieving comprehensive autonomic behaviors are remaining open, with the requirements of new technology and industry standards, the journey of automatic computing technology research has just begun.

2.2 Cloud

2.2.1 What is a "cloud"?

A Cloud is a new generation of commercial infrastructure that provides pools of computing resources (i.e. hardware, networks, storage and applications which delivered as services) and web interfaces which implements cloud computing.

Recently, the cloud service is becoming increasingly popular for business applications [23]. Basically, the cloud-based service is to deliver software, infrastructure, and storage as the services. Over the last years, cloud service has attracted many commercial organizations to move their sites to the data centers that shared pools of compute and storage, rather than having their servers locally. Examples of such leading cloud service providers are Amazon EC2, Rackspace, Google Compute Engine, Windows Azure, HP etc.

The cloud service has characteristics that differentiate it from traditional hosting. It provides the service as "On-demand self-service", gives user the ability to choose the CPU, memory, storage and network connectivity based on their demands. All computing resources are pooled and available over the network and accessible via various client devices (i.e. PC, tablet, mobile phone) [24].

Another important characteristic of cloud service is the rapid elasticity. Virtual resources can be dynamically provisioned and released with small management effort. Via cloud servers [25], almost infinite server power is presenting to the users.

The services are available and delivered to users whenever they need and wherever they are. In the cloud, clients scale services up and down in minutes, according to requirements change.

In addition, clients can readily take the advantage of pay-as-you-use basis model of cloud service. Business costs are claimed to be reduced by renting the infrastructures from cloud service provider [26]. In the cloud, you get the benefits from reducing the costs of maintaining multiple hardware facilities, purchasing datacenter floor space, as well as the costs of software upgrades and the human resources to manage it. The amount of cost only reflects the actual consumption of cloud resources.

With the benefits in cost saving, reliability and elasticity, cloud-based service has completely changed the way that companies to service their customers.

2.2.2 A cloud can be either public or private

A public cloud provides applications, storage, and other resources which available to the general public or a large organization with sharing same physical machines. These services are free or offered on a pay-per-use model.

The private cloud refers to internal data centers that within a company or dedicated physical servers housed in the cloud provider's data center. Without sharing cloud infrastructure with other users, the private cloud could be used to store site's sensitive information such as credit card, customer information and other private data.

Typically, cloud service is divided into three categories: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS).

Software as a Service (SaaS): In the SaaS model, which is referred to "on-demand software", the service is to provide access to business-relevant processes and various applications which running on the cloud platform, Such as virtual desktop, e-mail, CRM, high-performance computing and games. Applications are preconfigured and taken care of by cloud service provider. By this way, all underlying infrastructures are abstracted away from the user. Customers do not require to focus on maintaining the hardware which the applications run on or maintaining the applications. All needed is to use it over internet and pay on a pay-per-use basis.
Examples of service providers that offers SaaS: Google Apps, Zoho, Microsoft Office 365, Onlive and GT Nexus.

Platform as a Service (PaaS): The PaaS model delivers application execution services. It provides a computing platform plus the black-box services which application developers can develop and run applications on top of the cloud platform. This might include development tools that to build, test and deploy applications. With the Paas service, spending time and resources to install and manage the components are simply not needed. Virtual environment, develop tools, database and Web server are all provided and set up by cloud provider.

Examples of service providers that offers PaaS: Google App Engine, AWS Elastic, Windows Azure Compute, EngineYard and OrangeScape [27].

Infrastructure as a Service (IaaS): This is the point of interest in this paper. The IaaS is the most basic cloud service model which is usually seen to provide to the clients a large number of physical and virtual machines and other compute resources such as processing, raw storage and networks. In this model, the provider only delivers the standardized infrastructure, usually virtual servers, where the client is able to run arbitrary applications [28].

Examples of service providers that offers IaaS: Amazon EC2, Rackspace, HP Cloud, Windows Azure Virtual Machines, Google Cloud Storage[27].

2.2.3 Amazon

The one of the most well-known major cloud service providers is Amazon Web Services (abbreviated AWS). A cloud provider is a company that offers infrastructures and services of cloud computing platform to other organizations or individuals over the internet.

AWS Officially launched in 2006 [29] and began to provide a variety of online cloud services for businesses, such as data storage and cloud-based application processing.

Amazon is one of the key cloud services providers to some of the biggest and well known companies such as Netflix, Pinterest and Dropbox. The main services are Amazon EC2 and Amazon S3. Amazon EC2 provides virtual computing environments with operating system, services and application platforms for the customer's application hosting. Amazon S3 enables storage in the cloud and provides a web interface for customer to manage their stored data from anywhere via the web. To-

day, Amazon Web Services provides more popular components such as Amazon RDS, Amazon SimpleDB and Amazon SQS etc. Other cloud providers include Cisco, Citrix, Google, Microsoft, Rackspace, and Verizon/Terremark.

2.2.4 AWS Use Cases

To describe every feature of Amazon EC2 is outside the scope of this thesis, the features of Amazon EC2 that are most relevant to DR will be introduced.

To start using AWS, you simply sign up for an account and launch your applications. Cloud resources can be handily launched and managed by AWS Management Console. The AWS platform provides a number of AWS solutions to help customers deploy their own applications on the cloud or use varied cloud services that offering by AWS. Some main services are as follows,

Application Hosting: Running with Software-as-a-Service (SaaS) model, Amazon Web Services offer services as Amazon EC2 and Amazon EBS to enable application vendors or other customers to host their existing applications on the Internet.

Backup and Storage: AWS provides data storage services as Amazon S3. Customers can scale AWS storage up and down as they need to store information, files, backup or use as their backup location for disaster recovery solutions.

Content Delivery: Amazon CloudFront service is used by any customers to distribute web-based content and resources, such as images, videos, applications, or other files, to end-users over the Internet with high data transfer speeds.

Databases: There are many database solutions have provided by AWS, including Amazon RDS, Amazon DynamoDB, Amazon SimpleDB and several relational databases that customers can manage on their own. The relational AMIs on Amazon EC2 and Amazon EBS provide complete control over instances.

E-Commerce Applications: Amazon Web Services offer a number of solutions such as Amazon FPS to help customers handle a secure dependable e-commerce website.

High Performance Computing: With the possible access to tremendous amount of compute power resources on the cloud, this service is mainly targeting industries users or science organizations that require high computing performance and high

bandwidth to process their highly resource consuming applications, such as accessing very large data sets across multiple virtual instances or massively parallel processing.

2.2.5 Amazon Disaster Recovery Solutions

As introduced in the previous chapter, there are a lot of benefits of using cloud computing, one of which is to use cloud computing to deal with disaster recovery, making it more effective and cost lower.

Below is a list of the options of cloud disaster recovery solution that provided by Amazon.

- Multiple Locations - Place instances in different Regions and in availability zones. Multiple locations can help to insulate failures from other zones [30].

- Elastic IP Addresses - Providing the ability to mask instance or availability zone failures by programmatically remapping the public IP address to any other instance in same account [30].

- Elastic Load Balancing - Detecting unhealthy instances within a pool and automatically reroutes traffic to healthy instances until the unhealthy instances have been restored [30].

- Backup and Restore Pattern - Taking backups of current systems in advance and restore system from backup in case of disaster[31].

- Pilot Light in Recovered Phase - Setting up the replicated core data set as standby solution. Switch over to the standby system in disaster phase [31].

2.3 Major Service Outages

According to the AWS Service Health Dashboard [32], outages at the data centers are almost happening every day. They may only affect one or two companies for a short time but when there are severe outages on the cloud, so many companies are

impacted.

In 24 December 2012, at 12:24 PM PST on Christmas Eve, Amazon Web services experienced an outrage at Amazon's cloud computing data centers in North Virginia that took many services down. It was the fourth outrage of the year in this data center. Netflix (one of most-prominent customer of Amazon) services in the US, as well as some in Canada and Latin America are suffering from this outage. The disruption lasted 23 hours and 41 minutes and coursed by human error as told [6].

In 22 October 2012, an outrage started as a small issue affected data centers in North Virginia and gradually affected big area of other parts of Amazon Web Services in North Virginia. Affected services including Reddit, Foursquare, Minecraft and Heroku, suffered outages. GitHub, imgur, Pocket, HipChat, Coursera and others are having problems. Amazon Web Services said on its status-of-service page that this outrage appears to be a network-related issue[33].

In June 29, 2012, Amazon's East-1 US datacenter have Knocked Offline by violent storms in Virginia. The outrage has shut down the service in Netflix, Pinterest and Instagram and took with it a large percentage of some of the most popular sites[7].

In June 14, 2012, some major sites such as Quora and HipChat are taken down. The data center in Virginia was unavailable which resulted in lots of errors and latencies in the US-EAST-1 Region. In addition, the outrage has impacted on Heroku too. Amazon's service indicated that there were power issues[8].

In August, 2011, Amazon and Microsoft cloud services hit by lightning strike. This disaster had damaged power supplies of Dublin data centers and led to server failures and caused downtime on a large number of EBS servers. Although Amazon have managed to bring service back after 48 hours but the effects on businesses could have been serious[9].

In April 21, 2011, this service outrage which considered one of the worst cloud service outages, knocked down big customers such as Reddit, Foursquare, Quora and others for as many as several days. This claimed to be a server problem in the Amazon's northern Virginia datacenter[10].

Although Amazon is fixing the problems that caused its latest outage, this won't be the last. It is obvious that many common events could lead to failures, such as application failures, power failures, and system overload. It brings to the users un-

expected loss of services, if these services are holding on critical applications, the failures on businesses process could be serious. There is a need to create effective DR plan and build reliable cross-region services on cloud platform, which is also the main objective of this paper.

2.4 Disaster Recovery

Disaster recovery (DR), which in this paper focuses on the IT or technology systems, is a procedures and policies an organization uses to prepare for and recovering from a natural or human-induced disaster and help reduce or avoid losses. This could be recovery from applications error, hardware failure, connectivity issue, or some other disasters of IT infrastructure.

Businesses today are facing increasing volumes of data to protect. Service continuity is a vital requirement of the smooth operation of many businesses. Disaster Recovery (hereafter, DR) plan is therefore employed to minimize the damage from service interruptions caused by IT system failures. DR plans come in various forms: traditional DR plan is based on resource backup such as tape backup, disk backup and WAN-based backup. If the primary site becomes unavailable, the backup one takes over with the most recently replicated data. And typically, there are a few options to place the backup sites: [34]

- Onsite: the primary site and the backup system are placed in same location.
- Co-location: the backup system is placed in a remote location, separated from primary site.
- Cloud: the backup system is placed in the cloud.

A main principle of a effective backup strategy is always separating the primary and standby sites physically. As suggested in a survey from Symantec, a considerable amount of disasters incurred were quite regional [35].

However, traditional DR solutions require high infrastructure costs since they need to setup not only an extra standby site but also a high speed network connection to it, in order to keep data on the primiary and backup always synchorized up. Since

the backup site needs everything that the primiary needs, it usually costs the same. Which means, to have a backup site in this way, the cost of the IT infrastructure is simply doubled.

Many small businesses are then limited to such DR plan because of the rather high financial cost, in fact, according to Eweek, around 40 percent of small businesses have no formal DR plan at all[36].

2.4.1 Benefits of Disaster Recovery with Cloud

Fortunately, one of the most significant advantages of the Cloud is its cost efficient yet rather powerful disaster recovery ability. First, with pay-as-you-use model offered by many Cloud services, the cost of setting up a DR plan is minimized.

Secondly, no ad-hoc network connection need to be set to Cloud, as the Cloud, of course, is connected to the Internet already. Finally, the cost of hardware is also divided among the users of the Cloud.

A detailed cost analysis could be found from Wood, which shows a significant cost reductions with Cloud based DR plans compared to traditional DR one.[37]

Another advantage of having DP in Cloud is the time spent on establishing a backup site can be dramatically reduced. Thanks to virtualization technology, the "backup site", together with operation system, applications, data and system status information can be packed in a piece of data trunk and sent over to the Cloud. Since the time to establish a backup site in a Cloud is so short (typically a matter of minutes), the backup site can even be established on demand and brought online whenever the disaster is detected.

Although Cloud based DR plan shows its great potential for modern business, the general steps of recovering from a disaster still require a certain level of human intervention. For the sake of efficiency, an automated recovery process is always favored.

2.5 Relevant Researches

Although the cloud greatly extends disaster recovery options, the researches on cloud based DR solutions are still nascent [17]. Current DR solutions either come

at very high costs, or only focus on better performance of data replication and response time of backup sites.[38].

The goal of most studies currently is to limit the service downtime during disaster recovery procedure by providing high data availability and failover capability from primary/backup data sources module [39]. The key concept here is to have multiple replications of the original system and distribute them into different geographic location as backups of the original system. [40].

At the storage level, a cloud based system which uses Storage Area Network (SAN) has been suggested by [34]. SAN has a number of high performance routing devices and could provide redundant links for the application to access backup data. But since the SAN requires a large number of routing equipments to keep the data transmission network stable, and in addition, high bandwidth to achieve the theoretical process response time, the hardware cost is relatively too high for small businesses.

Caraman,M's paper[41] introduced a novel technique of building an end-to-end Disaster Tolerance (DT) enablement solution for IaaS clouds. Disaster tolerance (DT) refers to the capability of a system to survive from a disaster and to return to function in a relatively short period of time. Modern DT researches are mainly based on Remus, a commodity HA solution implemented in the virtualization layer [42, 43, 41]. Caraman,M's solution includes a seven-stage DT algorithm which covers all layers of IaaS clouds, however, it is still following the traditional DR plan, which involves two data centers, the primary is running and the backup is idle. The solution provides a full disk replication algorithm which adds the disaster tolerance on the fly by replicating the existing disk state together with the new disk writes. A strict requirement for this solution is a high speed connection between the data centers. Shriram Rajagopalan [43] presents an implementation of SecondSite. This solution extends the Remus module of high availability system by allowing a number of data centers to be replicated across wide-area Internet links.

Another approach to better data replication performance is pipelined synchronous replication [44]. This approach addresses the performance decrease caused by WAN replication latency. It tracks the replication with multi-tier servers. The applications record the consequences of the disk modifications and persist to a recovery site.

Nonetheless, none of these solutions has proposed an autonomic and dynamic dis-

aster recovery approach as introduced in this thesis. The key word of researches above is always geographical redundancy, which focus on creating redundant backup sites in cloud system that are physically separated in terms of location. The concept of geographical redundancy is used in this thesis too, but with a dynamic choice location model, it can provide less manual intervention and higher reliability.

Chapter 3

Approach

Processes and methods used in the research are explained in this chapter.

3.1 The Main Idea of the Design

The design is based on a set of real life cases of cloud based services. A prototype, which is able to create and maintain a large redundant system, is developed to assess the features mentioned in the problem statement. This research is mainly divided to two parts, creating a model which addresses the problems in the problem statement and examining the model behavior.

3.1.1 Formalization

Before presenting the solution, the principles and the design behind it should be described comprehensively. The design can be explained in multiple ways, a text-based documentation with diagrams are widely used. Texts could define all the concepts and functions with precision and details, while diagrams, often as complements, present high level architecture, complex relations and flows in an intuitive way.

UML (Unified Modeling Language) diagrams, component diagrams and BRIC (Basic Representation of Interactive Components) are two of the most formal ways to describe system architectures. UML is capable of defining components relationship, process flows and objects. It focuses especially on defining and visualizing the relationships between components, while the BRIC models the communication and interaction between components. The BRIC models, as a result, are especially

good at presenting the information about the event flow between multi-agents.

To create and maintain a large redundant system, which is the main objective in this research, the amount of sites could be massive and each site will have same status and similar behavior. Consequently, it would make more sense to focus on the behavior of individual sites instead of the interactions between them, therefore UML model seems to fit better.

Another tool to define the process or design program logic is Pseudocode. It is an alternative way to suggest some key principles of algorithm in this design. Pseudocode is not a formal program language by all means, but it resembles the logic behind it. It is easy to write and understand. It also can be very abstract when it's necessary, while conventional program languages often fail in this regard because they almost always inevitably have to give too many details due their strict syntactic rules.

3.2 Experimental Scenarios

In order to prove the functionality stated in the problem statement, some experimental scenarios need to be established.

To establish sensible scenarios, the first step is to work though the possible disasters and failures. After determining the disasters and failures, a set of scenarios can be established according to their impacts.

According to the principle stated above, some basic scenarios that cover most likely situations are built as follows,

3.2.1 Recovery When

Scenarios:

S1 Servers down: One or several servers of a site cease to function.

S2 Sites down: One or several sites in a zone cease to function.

S3 Zones Down: One or several zones cease to function.

S4 A site/zone is isolated by other sites/zones.

S5 A server has been compromised or malfunctioned.

3.3 The Objective

The ultimate objective of this research is to design a solution that is capable of surviving any cloud disasters. An optimal disaster recovery design solution should also take the initial cost, the cost of data traffic between sites and other factors that might influence the performance into consideration. However, such ideal objective and relevant discusses that presented later can only serve as a guideline to evaluate the prototype of the solution, for it is unrealistic to foresee all possible disasters and the complexity of real life scenarios could never be underestimated.

The design in this research then is considered complete when it covers all the features mentioned in the problem statement. The prototype is completed when it could demonstrate the functions to allow automatic recoveries from most common failures in the cloud.

3.4 Implementation Approach

3.4.1 Utilizing Existed Tools

To implement the software framework of DR solution is an important part of the prototyping. One way of doing it is to start from scratch. Writing everything from the start could indeed give the ultimate flexibility, the penalty behind it however is the potential huge workload, many pitfalls and great complexity of a big system.

Reinventing a wheel is always not necessary. Instead, build a system based on a set of already existed, wide used and quality tools seems to be a better strategy for this thesis because of the following two reasons:

- Quality tools in general contribute to the stability and quality of the whole system, and simplify the whole software development.

- System administrators are familiar with such tools, which lead to a flat learning curve when they adapt to the system.

3.4.2 Sandbox test environment or real life running

The prototype is designed to prove the concept, and assess all the features in the problem statement as mentioned. Despite of being capable of behaving like a real DP solution, it's indeed far away from a real commercial quality product hence it's not a option to even do test runs in real business.

It makes more sense to deploy/test in a sandbox environment. In a sandbox, everything is contained, tested and simulated. Failures happen in a sandbox will never affect anything outside. Therefore all kinds of tests and failures, no matter how insanely risky they seem to be, can be performed/simulated safely and the results can be easily observed.

However, after all rounds of test runs/adjustment iterations, the prototype should be a sufficiently good start point of a real product. Statistics data gathered during the test runs, experience in the prototype development will also be very valuable for the real product development.

3.4.3 Sandbox test environment - Local or Cloud

The other key factor that affects the testing environment is the choice of platform that it runs on. Run a test in a localized test environment has the advantage of creating a clean system with minimal external influences, hence, a real "sandbox" is easy and feasible. A local application is also much easier to debug compared the one that is hosted in cloud. However, the downside is that, since the thesis is all about handling disasters in a cloud, it simply cannot simulate the target environment with accuracy.

At this point, to ensure that the result of simulation is applicable to the real life, the decision favors more to cloud environment. There are many cloud providers on the global market today. But it has no great differences between different providers and services. Although the specific details differ, cloud providers provide similar tools and APIs (Application Programming Interface).

Amazon EC2 platform will be used in this research since it is the one of most widely used cloud platforms. The other important reason behind the choice is that it supports a wide range of operating systems (Explanation in chapter 2.1.3), plus it's able to deliver sites in multiple geographic regions throughout US East and Asia Pacific (Explanation chapter 2.1.5). In order to avoid the regional disasters, such like fires, tornados or flood, the ability of deploying the sites cross regions is

one of the fundamental requirement of the design.

3.5 Requirements of the Design

A set of experiments will be carried out to test/verifies features that stated in the problem statement. Prior to experiments, the prototype should have built a full redundant system with infrastructure and applications, running in parallel over a set of specified geographically separated locations. Due to the short time frame of this thesis, the prototype tested will not cover all functionality related to the discussion of whole design, instead, the scope will be narrowed down to cover only a subsection as a proof of the concept.

Hence, the experimental scenario will only consider those factors that impact the functions related to the problem statement. The testing is considered complete when it has enough data to conclude whether the solution has met the designated requirements.

3.5.1 Design of Experiments

The major experimental scenario is to test the ability of automated regeneration. A "regeneration" here is a process that whenever a site of whole system in suffers from power off, crashes or a shut down hence became unavailable, a replica is then established to take over the dead site after a certain period of time and the unavailable site will be removed (from the cloud).

This mechanism must ensure that after a failure occurs, the new site can be created that replicates the dead one, which means, the new site should have the same content and state. This requires an efficient way to ensure all relevant state is transferred to other sites in a timely fashion.

After data is transferred, the status report should be retrieved and stored up. These data are both addressing the basic requirements to the data availability of the services running on, and more important, as mentioned above, it is the fundamental factor to ensure the stability of entire system in the long run.

During the operation of service on the sites, a report of the number of running sites and their locations should be generated regularly, as well as an event report which indicates the status of site regeneration during the recovery process. Equally, it is

essential to monitor the service continuity during the recovery process. This is the fundamental measurement which indicates whether the stability of an online service is influenced by an ongoing disaster.

3.5.2 Expected Results

In advance to complete the research, there are some expected outcomes. These outcomes will be evaluated in relation to the stated questions in the first chapter. In general the desired result is the automatic recovery from failure with the needed functionality proved through the specific scenarios. In addition to when handling the state in the face of failures, applications should be able to resume in acceptable timeframe. The entire system is up and running after failure and regeneration with data consistency. It's also expected for the system to have a sound service continuity during the event of failures.

Chapter 4

Results: Design

4.1 Overview of the Design

In order to fulfill the functions stated in the problem statement. The architecture of the whole design is described in following chapters.

Some terms will be used throughout the coming chapters. The short explanations of them have been given as following:

Hydra

This is the name of the whole system designed in this project.

This name is inspired by a beast in ancient Greek mythology. The Hydra is a water beast that possessed many heads, as mentioned by poets, once a head cut off, two more would grow back. The same goes for this design. If a site in a zone fails, another replica will simply be rebuilt somewhere.

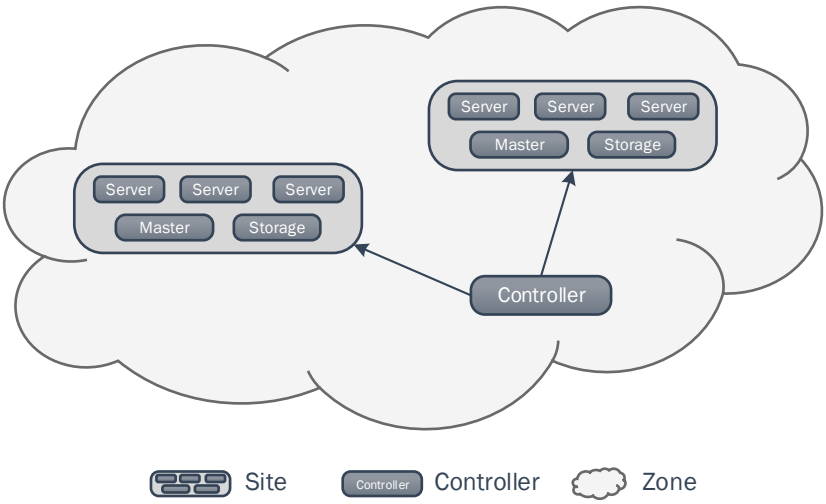


Figure 4.1: Key Components and Concepts in the Architecture. A Controller is monitoring two sites in a zone.

Zone

A *zone* includes all sites that located in the same Availability Region (Explanation chapter 2.1.3).

Core System

A set of servers in a site that is capable of providing *Service* to users. It normally includes a master server and several agent servers which have different functions.

Service

The *Service*, which will always start with capital "S" in this paper, refers particularly to the customer service that hydra needs to provide. The *Service* will be hosted on a *core system* in the site, for example, it can be Web service, file storage service etc.

Controller Server

A server serves as a controller node in hydra system. The controller server is capable of:

- Supervising servers in a site and managing their responsibilities.
- Communicating with other controllers in the same *zone*.
- Detecting if another site is gone and then issuing a rebuilding process.

Root Controller

It represents one of the controller servers which has the extra responsibility of

- Communicating to all *zone*s.
- Detecting if any *zone* is gone and then rebuilding the *zone* in another available region.

Master Server

A server that is responsible of management in the *core system*. The main responsibility of master server is to control the behavior of agent servers. It should make sure that the *Service* hosted on the agent servers are running, and, in the same time, it ensures the agent servers follow behaviors predefined in their configuration files. Status reports of agent servers will be generated by the master server and sent to the controller regularly.

Agent Servers

Servers in the *core system* that are responsible of hosting a set of *Services*. Behaviors of agent servers are controlled by the master server within same *core system*.

Site

The basic unit of Hydra. Every *site* should include a controller and a *core system* which provides the *Services* of the site.

Common industry terms for disaster planning:

Recovery Time Objective (RTO) [37]: The duration of time that a recovery takes from the interrupted service.

Recovery Point Objective (RPO)[37]: The most recent backup point that prior to any disaster.

4.2 The Architecture

4.2.1 Platform and Environment

The design in this thesis provides the function of creating large redundant sites system which across several Regions. The Hydra system keeps tracking the status of every single site and runs the recovery procedure from an unforeseen disaster by destroying/rebuilding the site/zone automatically and effectively.

Here follows a description of the whole architecture used throughout this paper and how it affects the experiments.

Cloud Platform

Cloud environment is natured for the disaster recovery. Without the capacity limitation of physical servers, in the event of a disaster, new resources can be quickly launched in cloud to ensure service continuity and short RTO. As mentioned in the previous chapter, Amazon cloud, the one of most widely used cloud platforms, is used as virtual environment in this design. The Amazon Web Services (abbreviated AWS) provides Amazon EC2 that enables creating and destroying of instances in a matter of minutes.

4.3 Scenarios

Possible scenarios for the model:

- Whole zone is down because of natural disaster.
- Disasters such as fires, tornados or flood might affect a whole Availability Region and result in a failure of whole zone area.
- Whole zone is isolated by other zones There might be a connectivity issues or high latency between zones. When a zone is functional but suddenly lost all connections from other zones.
- Several servers fail because of hardware/power problems. The failures of hardware or power equipments might lead to partial outages in a zone.
- Connectivity issues inside a zone Every Availability Region consists of many internal data center facilities. There could be outrage of networking

equipments in a datacenter and disconnects between controller and master or master and servers in a zone.

- A server which is part of a site fails A Web server ceases to function. This might be caused by malfunctioned VM or (DDoS) attacks.
- A server has been compromised or has abnormal function. A controller or A server is still partial functional but suspected to be compromised.
- A storage server fails A storage server in a site ceases to function, or by any reason has been compromised or has lost data.
- A site master fails A site master is suffering from application failure or cease to function completely.
- A controller fails A controller/root controller ceases to function.

4.4 Architecture

The overall architecture of hydra system will be described thoroughly in this section.

4.4.1 The Hydra

The architecture presented is focused on to have features of automated, adaptable and reliable as stated in the problem statement.

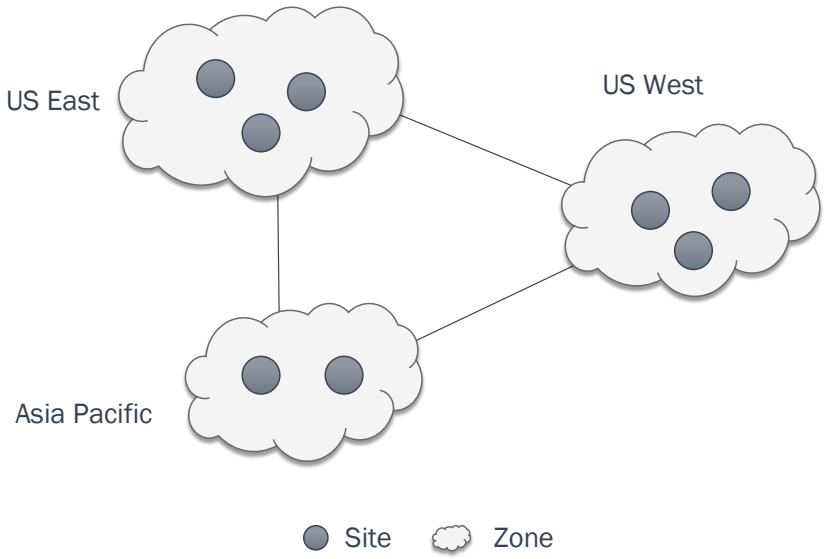


Figure 4.2: The Structure of Hydra. The Hydra combines several sites which running across multiple zones.

With a more abstract view, Figure 4.2, shows that the Hydra combines several sites which running over multiple zones. The zone located in the physically distinct regions to ensure high reliability. The decision of the zone's location is made by choosing the region as close of the main service users as possible. The zone may include one or more sites with controller according to business requirement. With fully functional structure, every controller has the ability to replicate itself and create the entire system without external intervention.

4.4.2 The Zone

Next sections are focused on explaining the different components of a zone.

The Core System and the Site

Developing the core system involves implementing the system according to the requirement of the scale of the service and performance. For the purpose of test, the operating system and the services running on is not essential, because the intention of this design is to capable of supporting any service with same architecture. In the demonstration, all servers and control nodes will use the minimal Ubuntu installation and supporting Web service.

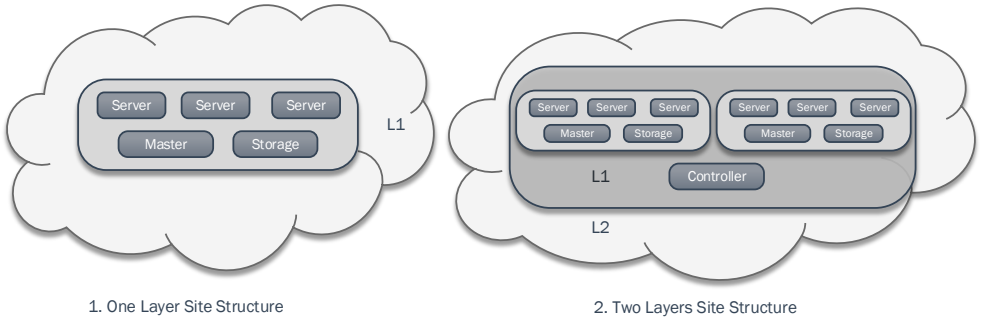


Figure 4.3: Possible Structure of Site. The left side is a traditional implementation with one layer structure. The right side is the structure with an additional layer.

Figure 4.3 illustrates the introduction of two optional structures of the site. The left side is a traditional implementation with one layer structure. The right side is the structure with an additional layer. The one layer structure includes a core system and a controller node. For the Web service to function properly, one or more web servers are needed, as well as a database server in the core system. The controller node has the puppet master installed on the master server in the L1 layer. These servers in core system and controller together define a site, which is a basic unit in the Hydra system. In this design, the master server of the L1 is responsible of controlling/ maintaining of the whole site and connecting with other sites.

The right side of the graph is two layer site structure, the first layer is almost same as single layer design. In L2 layer, a controller has introduced as a second master. The controller has also the puppet master installed and manages all servers in L1

layer as puppet clients. The responsibility of the controller is to ensure the running state of L1 layer and communicate with other site's controllers. This will open the possibilities for the site to host different types of service in the node with only change the behavior of servers in L1 layer. User can create his own node solution that fulfils the L1 structure. The controller will operate as before and the cloud will therefore behave the same way.

The L2 layer structure has been chosen in this project because the flexible attribute. The sites can easily adapt different requirement such that the architectures independent from content type they provide.

However, the weakness for this solution is with a new controller layer introduced in the site, the implementation becomes a little more complicated and it costs more with extra instances needed.

Cluster Storage in the Site

There are two main approaches when transferring data: synchronous and asynchronous. With synchronous transferring, data is immediately updated in multiple locations whenever they have been changed. This sets a high demand on network performance and availability. With asynchronous transferring, data is not updated right away after changes. It is transferred while the network has relatively less traffic. Many database support asynchronous data replication, it can work with lower bandwidth and easy to deploy. This is acceptable in many scenarios such as a non-active backup site.

However, an essential requirement for this project is a decentralized storage that keeps tracking the changes in the databases and synchronizes timely with all storage servers. The RPO (Explanation in chapter 4.1) is practically a business decision. In order to avoid data losses in the case of disaster, real time data synchronization of changes in every single site is particularly important. The ideal strategy is to synchronize data whenever it has been modified. Keeping database relatively up-to-date during continuous modification is crucial in cases where aggressive RTO needs to be met. However, for some servers, there is a large number of data that need to be replicated over a long distance network. The components involved in data transmission must be extremely reliable to avoid possible overload failures. Apparently, frequently updates are impractical since it greatly increases the burden on the server and network. The compromise is to choose proper synchronization interval according to the acceptable data loss of the services.

In the design, the controllers are sharing a directory which contains templates of new hosts in the system. Any future modification or upgrade on the configuration file of new host is easily to implement on one controller, and then be automatically synchronized to all controllers in a short interval. In the same way, a storage server is hosting cluster file system in every site that will responsible to synchronize Service data. The database replication ensured the data availability, since any single site is a duplication of others.

4.4.3 The Structure of the Zone

There are three possible approaches to provide a functional structure solution for the zone. The advantages and disadvantages for them are presented, giving an overview of how the components involved are interacting with each other. And then explains why the finial prototype has been created based on the previous discussions.

Possible Approaches

Approach 1: Multi-controller Solution

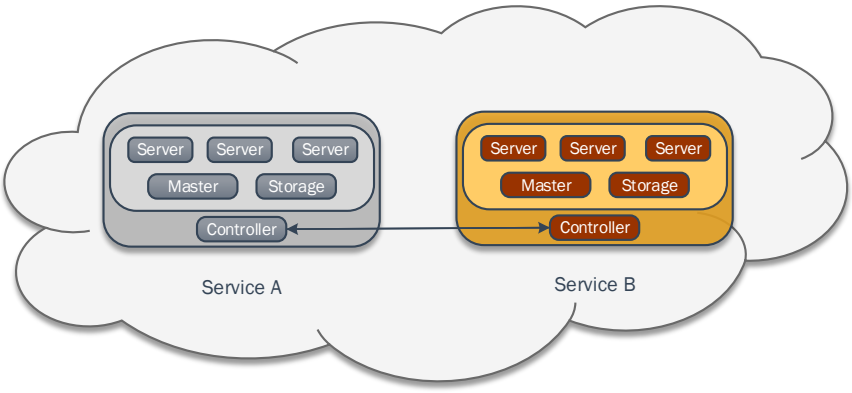


Figure 4.4: Site's Structure of Approach 1. In this approach, controller exists in each site. Different Service A and B are hosted by two sites in same zone respectively.

Structure

This is the most reliable and robust model. In this solution, a site and its controller always present as a set. As showing in figure 4.4, a controller, which working as a hypervisor, will appear in the zone together with its corresponding site. The main responsibility for the controller is to ensure all servers are functioning in the single site. With only few of servers in one site, when the time comes for recovery, multiple sites could trigger the reconstructions simultaneously, and the controller could then rapidly provision a full scale site.

Storage

The database synchronization between every site is crucial. Every single storage server is a complete duplication of others. Each site encompasses the database of different Services (Explanation in chapter 4.1) in the storage server, as well as the data repository for the building process of all servers stored in the controller. The purpose of this setup is to construct a massive redundancy system, which could further increase the availability of the Services, and more important, it gives the biggest advantage to this solution, the robustness. Even if there is only one site exists, this single site will has enough information and ability to replicate itself and then create the entire Hydra system.

With all the databases available from any storage, the structure is independent to the Services running. The procedure for creating servers for different Service is differentiated by using predefined EC2 templates.

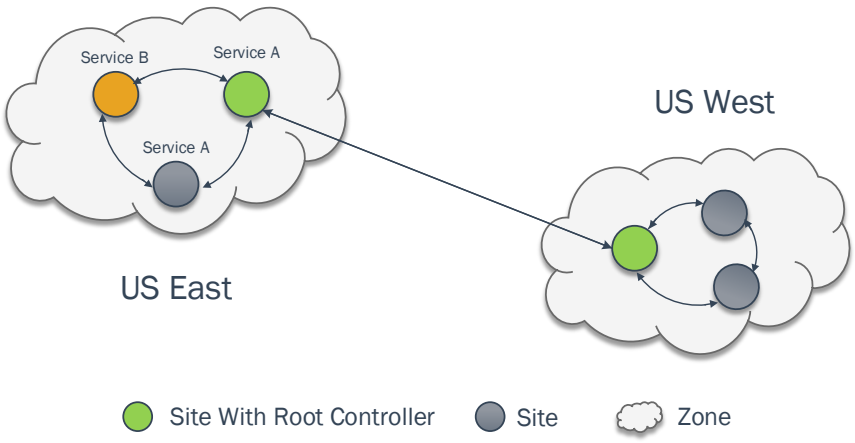


Figure 4.5: Communication Method for Approach 1. Root controller in US East is communicating with root controller in US West.

Communication

The Figure 4.5 illustrates the communication method inside a zone and between zones. The controller has the responsibility to communicate with each other directly in the zone. This strategy that every site is aware of every other one ensures the continuous detection of availability of every site. The same communication algorithm can also be expanded between zones. In the zone scope, a controller per zone is designated as root controller to interact with other zones. When any root controller becomes unavailable, another controller in same zone will temporarily take over and keep contacting with other zones. In order to provide better reliability, in the meantime, the original root controller will be rebuilt.

Scalability

The most significant feature of this solution is the unified structure. The system can easily grow and scale since every site has same infrastructure and use same algorithm. The new site and the controller node are fully duplicated from the current site and hence inherit all properties and database. The procedure for scale up the number of sites is to edit the entry of alive list (Chapter 5.2.3) in every controller. Then, new site will be considered "missing" and be created automatically.

Disadvantages

Apart from the actually servers that provide services, many controller servers are needed in this solution. Those controllers require additional cost and might cause financial difficulties to smaller businesses. The other complication caused by this complex structure is the large amount of data traffic spent on interaction and synchronization, which could contribute to more potential traffic overheads and system errors.

Approach 2: Single-Controller per Zone

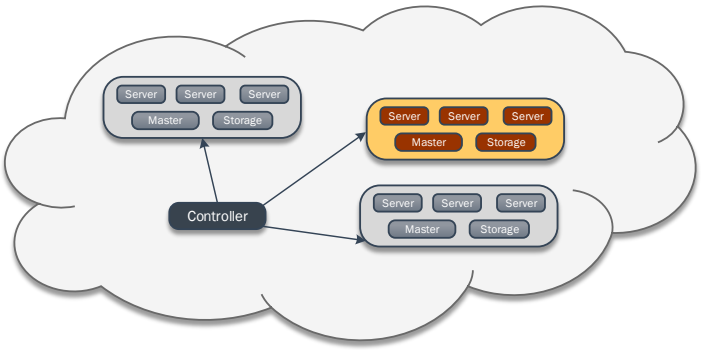


Figure 4.6: Site's Structure of Approach 2.All sites located in one zone are directly monitored by same controller.

Structure

This multi-site solution reduces the number of the controllers in each zone to only one. All sites located in one zone are directly monitored by same controller. This controller is working as both zone controller and root controller. The decrease in the number of controllers simplifies the structure and reduces the traffic for the interaction between sites.

Storage

The database redundancy for the Service is as same as approach 1. Each site has the copy of data for all Services and they are always synchronized up. But the data that is needed to create a new server, which is kept in controller, is hence decreased to only one copy per zone.

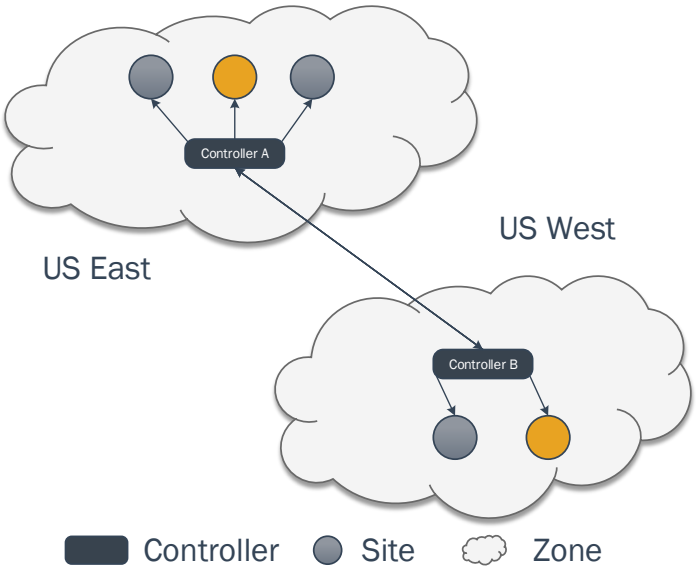


Figure 4.7: Communication Method for Approach 2. Controller in US East is communicating with controller in US West.

Communication

Besides controlling of sites, the controller is responsible to communicate with controllers in other zones too. In other words, the interaction between sites is no more required. This solution has lightened the burden of communication traffics inside zone.

Disadvantages

The key consideration is the greatly increased pressure of sites managing placed on the single controller. The other is the system resizing. Having the controller directly control every site in the zone, the increasing and decreasing of sites will have to be implemented by editing the configure files of Configuration Management Software on the controller. And one more thing to notice is that in the case the controller is not reachable, it is apparent hard to assess if this failure caused by the single instance or whole zone.

Approach 3: Single-Controller per Service

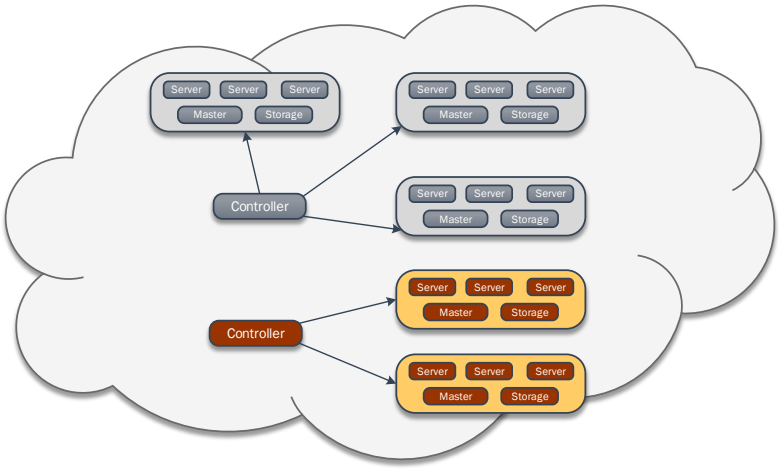


Figure 4.8: Site's Structure of Approach 3. Only the sites assigned to host same Service are controlled by same controller.

Structure

Inspired by the merits and demerits from two approaches above, the approach 3 has developed a balanced solution to limit the number of controllers according to the Services. As illustrated in the Figure 4.8, this solution is quite similar to the previous one, but a rule of correspondence established between the numbers of controllers in each zone and the number of different Services. The sites which are assigned to host same Service are controlled by same controller.

Storage

The storage server is no longer containing or synchronizing the database unrelated to its site's holding Service. The synchronous procedure and the data traffics have hence dramatically decreased. However, this also decreases the level of data redundancy for the whole system. To keep the advantage of multi-location, the sites that serve for the same Service must be presented in more than two zones. It designed to have no primary storage with any zone. The data load is distributed evenly to the sites across multi-Region. This kind of distribution is to against service disruption or data loss occurred in whole zone.

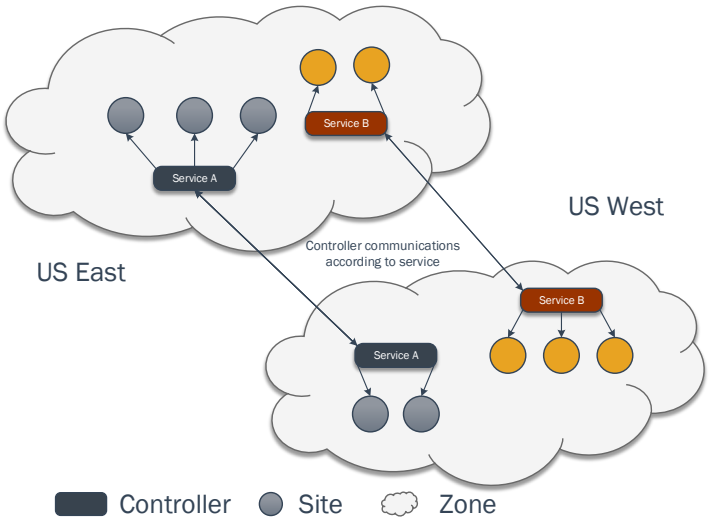


Figure 4.9: Communication Method for Approach 3. Only controllers serve for same Service are able to communicate with each other.

Communication

Although each zone could have more than one controller, there is no communication between controllers belong to different Services. The only communication between controllers must cross the zone with controllers serve for same Service. Based on this concept, this solution can also be understood as the hydra system handles only one type of Service. This can also be expanded to different service, adding another Service is construction of a new hydra.

Summary

It is apparent from above solutions that the approach 1 is the most robust yet the most costly and complicated design. The approach 2 has less controller and interactions between them, however the downside is the heavy workload on the controllers and the flexibility is somewhat lower.

For above reasons, approach 3 will be used to develop finial prototype in this thesis, it has increased the utilization of controller without the potential overhead. In addition, it has further decreased the complexity by supporting only one type Service. Although it's suboptimal in system resizing and dealing with the controller failure, this thesis mainly focuses on only the disaster on the cloud, failures that will influence whole data center. At this point, this approach can be an attractive alternative

for companies that require modest scale sites without pressing need of auto scaling.

As discussed in above sections, the differences of the three approaches are summarized in the table 4.1.

Category	**Approach 1**	**Approach 2**	**Approach 3**
Architecture	Complicated	Simple	Simple
Complexity of Algorithm	High	Low	Low
Data Synchronous	Low	High	Medium
Synchronization Method	Complicated	Medium	Simple
Scalability	High	Low	Medium
Recovery Rate	High	Low	Medium
Cost	High	Low	Medium

Table 4.1: Summary

4.5 Behavior

4.5.1 Controller Server

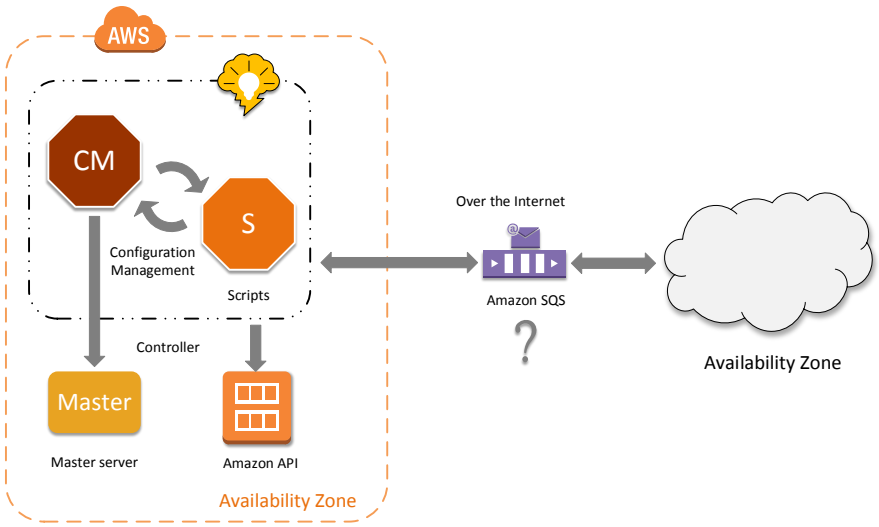


Figure 4.10: Structure and Behavior of a Controller Server

Figure 4.10 has abstracted the infrastructure of a controller server. Acting as a "brain" in the whole hydra system, controllers need to have the ability of making decisions about the right reactions to potential situations. This "intelligent" part is achieved by *scripts* that keep running in the system. During every execution, the *scripts* will fulfill the jobs such as obtaining updated information of all hosts in local site, communicating with other controllers, terminating hosts that do not work properly, and creating new hosts. In order to obtain the updated status of hosts and implement termination/creation, the script is designed to do these maintenances by accessing cloud services interface such as *Amazon API*. The Amazon API is a combination of tools that provides programmatic access to the Amazon EC2 web service.

Another major part in the controller sever is *Configuration Management* software, it ensures the continuous running state of *script*. In the mean time, the behavior of emphConfiguration Management software is controlled by the*scripts* too. The *Configuration Management* software is responsible of ensuring designated configurations being applied on the new controllers and masters generated by this controller sever, and enabling the configuration files editing while the running time of hydra system. It needs also to support an interface that allow human administrators

to enter new policies to update the operation of the system. Changes and instructions received from *script* will be pushed into local master servers timely on the fly. This function is required for further growth of system. Adjusting system to varying circumstances automatically and preparing it to future demand.

On top of that, developing a communication method between controllers in different zones is another key consideration, as displayed in the middle of the figure, using the *Amazon SQS* is one way to solve this. Amazon SQS (Amazon Simple Queue Service) works as a distributed messaging queue service that provides a high available authenticated message system. It is relatively easy to build an automated and secure workflow working with Amazon EC2 and other AWS infrastructure services by using Amazon SQS, however, the penalty of this choice is it limits the cloud platform of whole hydra design to Amazon. For this reason, the communication method needs to be independent from specific cloud form. An alternative choice is to use general client-server communication protocols such as HTTP protocol. The feasible communication methods according to particular cases are discussed in the next sections.

4.5.2 Master Server

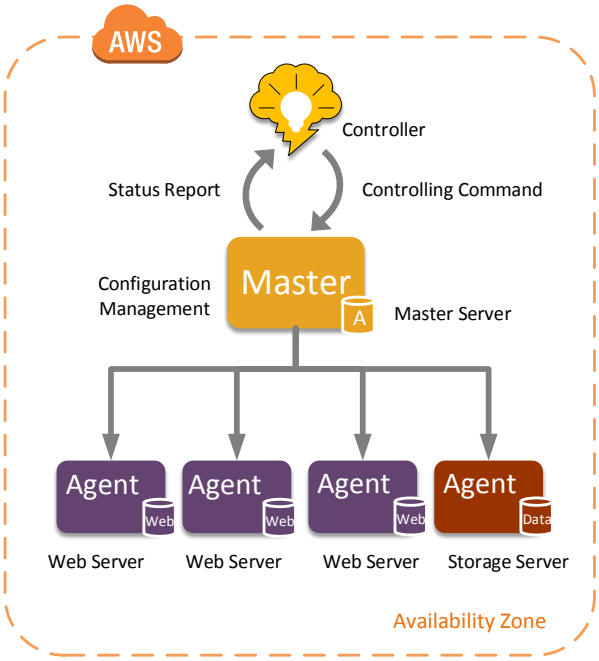


Figure 4.11: Structure and Behavior of a Master Server

The architecture of a single site is illustrated by Figure 4.11. *Master* server is a "head" of the site in a zone. It has both the server and client parts of Configuration Management software installed. As outlined in chapter 4.4.2, purpose of this design is to ensure the *master* server acting as an agent of local controller server to implement all instructions delivered by the controller, and then working as a master server of Web servers in the site to control the behaviors of Web servers to specific services they provided accordingly. In addition, detailed reports of operating status will hence send back to controller server from *master* server regularly. Those reports will be analyzed in the controller as a measurement of rebuilding decision.

4.5.3 Communication Mode

In order to automate the disaster detecting and self-healing process, the architecture needs a way to communication between controllers, sites and zones.

Communication between the EC2 instances in different clouds is commonly handled by using a existing models of distributed queue or messaging service that pre-

sented for could services, such as Amazon SQS, which is exposed with a friendly interface that fits for cloud applications. The queuing service or messaging service can fulfill some additional guarantees, such as delivery of synchronous messages. This may be the necessity of some communications method needed by many programs. Nonetheless, the use of message queue application that particularly designed for Amazon web services will limit the hydra system to certain cloud platform, which decreased the future development ability of the hydra system.

For the options of common communication protocols, text protocols are the first one to be presented. It has the simplest implementation and also very easy to control and programming. To communication in ASCII format, is the most common choice because it is the simplest to implement and the most portable. If a protocol becomes complicated, it may in the same time become difficult to implement. However, for the protocols such as creating socket connection by scripts, the time interval between requests and responses is hard to control, the port that accept the signal is needed to open and listen continuous during the conversation. In addition, the conversation is in particular fragile to the conversation hijacking by hacker or middle ware.

Another option is Web services, one of the reliable and secure communication solutions. Web services are a standardized way to use web applications that communicate with each other for the purpose of exchanging data. The interest in using Web services is that it is independent of operating system and machine architecture. It allows different applications to communicate with each other despite of which source they come from. It makes the hydra system more applicable to different operating system. The Web service can be added into a web page or a program and easily send requests from browsers and to return the messages on the requested pages. In order to protect the communication between sender and receiver, an authenticate protocol: SSL/TLS protocol needs to be included.

Other options such as OAuth can also be taken into consideration. OAuth is an open protocol for interacting with protected data of an HTTP service. As a standard protocol, it uses authorization in standard method from web and other applications and enhanced the security of interaction between controllers over the network.

4.5.4 Security Requirements

Hosting sites on the public cloud platform, all databases are not protected behind the firewall of corporate intranet. Instead, they are storing on the physical machines

which sharing with other users. This will always raise security concerns. The relationship between user and provider is more complex in the cloud environment. There is a need to ensure the security of the data in the controller node and storage, as well as a generally security mechanisms for the end-to-end data transactions between servers.

The basic security requirements are described below,

Authentication Mechanisms

The operating mechanism of hydra system is depending on the correct interactions between controllers. One controller will take actions accordingly to instructions they have received from other controllers. The interaction between them requires the verification of the identities of the sender and receiver. The type of authentication mechanism that has to be used depends on the trust model. Several methods can be used to authenticate services, one which is referred to as a digital signature, the other include message authentication code (MAC), password-based encryption methods and certificates.

Authorization Mechanisms

Every controller in the system is sharing a storage which contains the synchronized information and configuring files about all hosts in the system. Accesses to such resources in the site must be authorized. This is important to ensure system resources are granted only with appropriate servers belong to the hydra system.

Data Integrity

Data integrity is needed to ensure that unauthorized modifications of data on the hosts or during transmission will be detecting and reported. If the host in the system has considered to be a compromised host, it should be terminated and rebuilt.

Chapter 5

Result : Implementation

5.1 Prototype

Based on the concepts discussed extensively in design chapter, a prototype has been developed. This prototype includes an implementation of the core system, a cluster file system, as well as a control layer and core system layer implementation.

Due to the time frame of this thesis, the prototype testing will not cover all possible scenarios during a disaster or system failure. Instead, the experimental scenario will only consider those factors that impact the functions related to the problem statement. Hence, after the choosing among various disaster or failure scenarios, some basic scenarios that cover most likely situation of server failures will be present in this chapter as a proof of whole concepts of hydra design.

The scenarios that will be demonstrated with prototype were chosen based on the different factors specified in the problem statement. Detailed explanation of every scenario and their recovery process are following:

5.1.1 Possible Scenarios

Scenario 1: A controller has shut down by a disaster

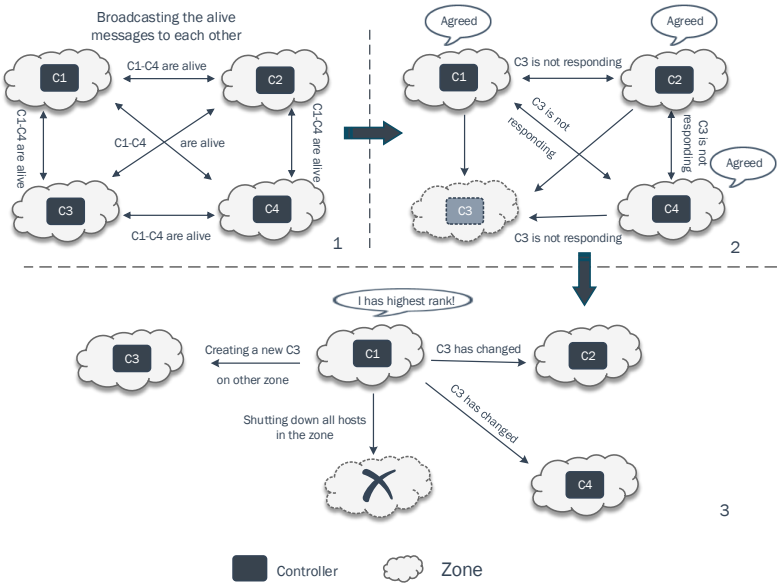


Figure 5.1: A Controller Has Been Shut Down by a Disaster. Controller 3 has been shut down due to a disaster occured in zone 3. Controller 1 starts the recovery process after the failure of controller 3 has been confirmed.

The scenario shown in figure 5.1 is illustrating a failure occurred related to a controller server. In this scenario, the hydra system experienced an outrage at cloud data center 3. The outrage was coursed by a disaster that has shut down big area of services in this data center. The controller 3 has been affected and become unavailable.

According to the actual outrage events indicated in background chapter 2.3 – *Major Service Outages* of recent years. There are several examples of severe outages that impacted wide spread cloud service in a data center. Some sites have been shut down for several hours to even several days. This scenario has simulated the situation of such outrage event on the cloud.

In the stage 1, this is normal stage where controllers in each zone are communicating with each other regularly. The "alive" messages of all controllers are propagated by each controller server at regular intervals.

In the stage 2, controller 3 has been impacted by the outrage and becomes unavailable. Other controllers have noticed this error. The regular status message of controller 3 becomes "not responding". After several failed attempts to connect with controller 3, other controllers confirmed that the status of controller 3 becomes unresponsive and agreed to start maintenance process.

In the stage 3, controller rank checking has been started. The aim of this checking is to find who has the highest rank. The one that wins will start maintenance process. This process is used to simplify the election process of choosing a proper controller server to do the maintenance and ensuring there is only one controller can take the job.

The controller 1 has won as well as the maintenance process has been started and proceeds as follows :

- Reading information of reachable instances in zone 3 and record them (the zone 3 is where the controller 3 is living).

- Terminating instances existed in the original zone 3 according to the record. If an instance is not reachable yet, it will be detected and terminated in the future when it becomes reachable.

- Getting the list of available regions (zones), the one of which has least controllers and near to the controller 3 will be selected.

- A new controller 3 will be created in the selected zone while the termination of original one. Maintenance of local zone will be started by new controller 3 after its regeneration (Process of local maintenance is described in the following scenario 5).

- Starting cluster file system maintenance. (Detailed process will be described in the following chapter)

- Updating information of new controller3 and other new instances belong to new controller 3. This information is synchronizing with all controllers.

- The maintenance process is finished and stage 1 is restarted.

Scenario 2: Isolated Controllers A

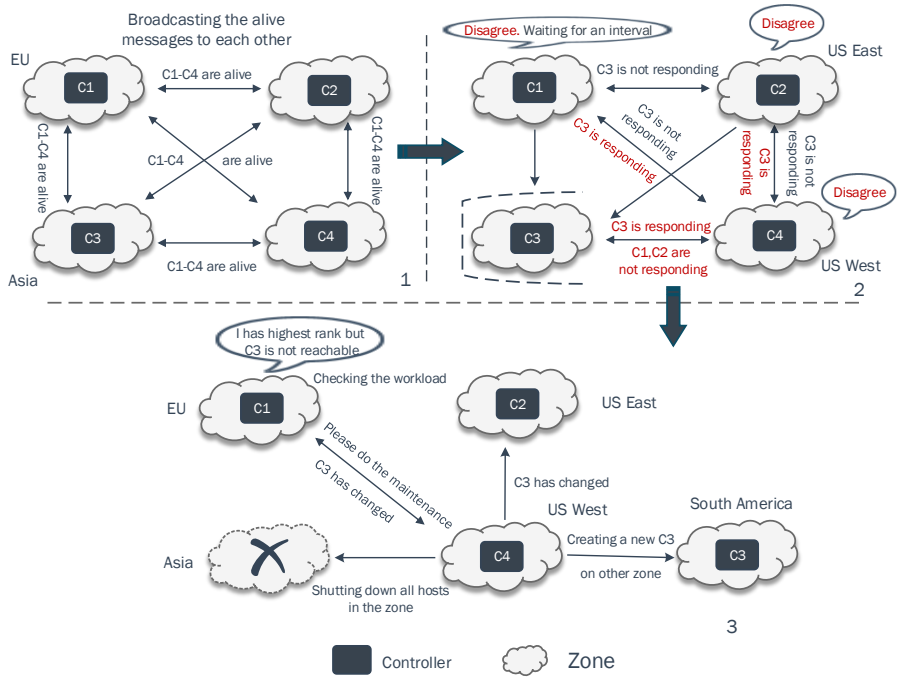


Figure 5.2: A Controller Has Been Isolated. Controller 3 has been isolated from EU and US East. After the failure on controller 3 has been confirmed, controller 4 starts recovery process, since it is the only controller that is able to communicate with controller 3.

This scenario emulates a network outrage that has suddenly cut down the connection between Asia and Europe. The controller 3, which located in the data center of Asia, is only capable of communicating with data center in US West.

In the stage 1, as usual, controllers located in different zones are functioning normally and broadcasting system status messages to each other regularly.

In the stage 2, the network connection between Asia and Europe has been damaged by a lightning strike. The connection between Asia and US East has also been impacted. Consequently, the controller 3 has been isolated from controllers in 1-2 zones.

During next status checking of hydra system, the controller 1-2 have noticed the outrage with controller 3 and then sent out "C3 is not responding" error message. In the mean time, with no responses received from 1-2 zones, controller 3 has sent out the "C1-2 are not responding" message too. However, a particular aspect about this case is the zone 4 is still able to communication with every zone. Thus, controller 4 is keep sending out "C1-C4 is alive" messages back to all other controllers continuously. At this point, the controller 1, which has the highest rank, has got the "disagree "result from the comparison of messages it received. It starts to wait for a certain interval and check the workload in the same time. The interest in this waiting and checking is by the assumption that the network problems could be occurred frequent in the real world. Normally it is not necessary to restore the whole zone since the network connectivity could be recovery in a short time. But there is a possibility that the zone has isolated from users it used to serve or it really is not working properly. In this case, the controller 1 will keep monitoring the changes of workloads on reachable controllers, and to determine if the controller 3 is still functioning. The recovery stage would be start if other controllers are suffering from rapid growth workloads. This strategy is important for ensuring the system performance while improving system stability.

In the stage 3, the controller 1 has found the increased workload went beyond the bound of preset value. The maintenance process is hence started. A difficulty arises from this stage is the controller 3 is actually unreachable from controller 1. To assist with this situation, the maintenance work is then passed from controller 1 to controller 4, which still has the communication with controller 3. With similar process in previous scenario, a new controller 3 will be created in a new zone while the termination process of entire previous zone 3.

Scenario 3: Isolated Controllers B

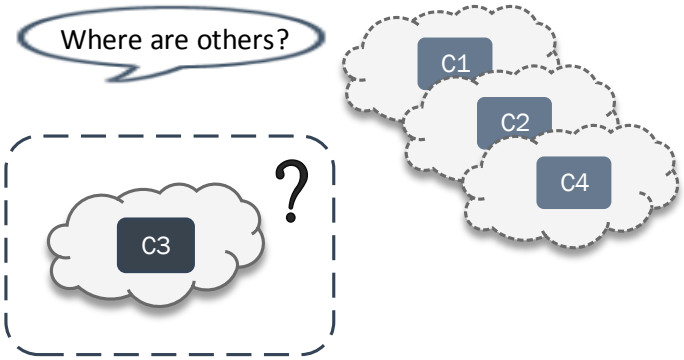


Figure 5.3: Completely Isolated Controller. Controller 3 has been isolated from hydra.

Based on this scenario, another challenging case should be mention here: what if controller 3 has been completely isolated by others? With the same mechanism, it will attempt to create a complete new hydra system somewhere else. One way to solve this issue is by using assumption that since a zone has been isolated, it is hence not able to execute the creation command on other zones. In this situation, there is a need to confirm that controllers should not create new controller in its same zone. This concept presents the basic requirement in any effective disaster recovery plan, which is the primary and backup sites must be geographically separated. This is also one of essential part of fundamental principles of hydra system.

Scenario 4: Isolated Controllers C

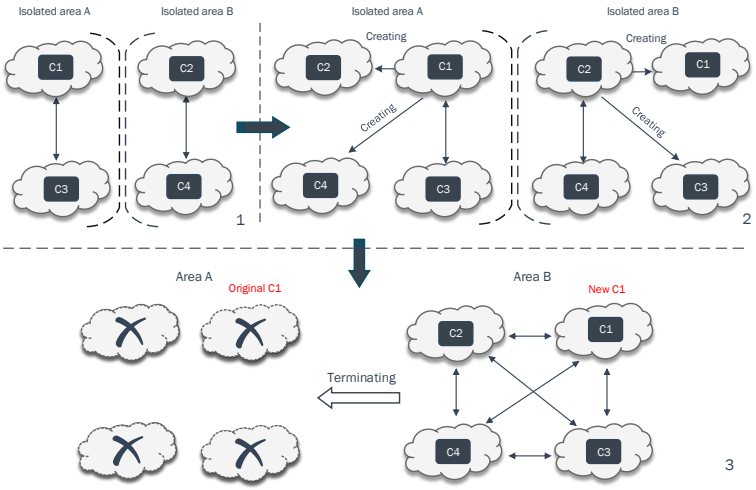


Figure 5.4: Isolated Controllers In Two Area. Hydra has been partly isolated. Two separated parts are created new hydra systems respectively. Hydra in area A has been removed after tow hydra found each other.

Nonetheless, there is still a possibility that controllers in two separated areas have created a new hydra system respectively. Or after the regeneration of new controllers that had been isolated, the corresponding ones appeared again after the recovery of network connectivity. To address this problem, the first step in maintenance process is to search for all instances in every available region that named by same hydra project. The newest controller that has the highest rank will continue the maintenance process in order to terminate other hydras in the first phase.

For example, as illustrated in figure 5.4, the controller 1,3 and controller 2,4 have been isolated to each other in the stage 1. With no connection from other side, the two systems have both decided to recover hydra system in their own area.

In the stage 2, since controller 1 and controller 2 have the highest rank in their own area, the recovery work is done by them respectively. In this stage, two separate hydra systems have been established and started to serve for their local users.

In the stage 3, the two hydra systems find each other after the network connection has been re-established. During the system status checking, the controllers send out the error message about duplication name of hydra hosts. As before, the maintenance process will be invoked according to the error message. The first step in

the maintenance process is still the rank checking, two controller 1s will find that they both have highest rank, and then to start launch time comparison. The launch time is a value that indicates the creation time of every instance in the cloud. With the mechanism designed in this project, it assumes that the newer instance has less error as well as up to date configurations. So whenever the launch time comparison happens, a newer one will win and be kept while the old one be terminated. Hence, in this stage, the original controller 1 will skip the maintenance process and be terminated by the new one. However, other consideration is arisen here: controller2,4 that generated by original controller 1 in Area A are newer than the original ones in Area B. What will happen then? Here comes another rule, before the termination of duplicated instances based on the launch time comparison, the controller that starts the maintenance will prior try to keep the instances that sharing the some cluster file system with it. This is also a guarantee of the stability in hydra system to avoid unnecessary changes of whole system.

By the mechanism explained above, we can see in figure 5.4 stage 3, the hydra in area A has been terminated and the one remained in area B is able to resume to normal status again.

Scenario 5: Inside Maintenance of a Zone

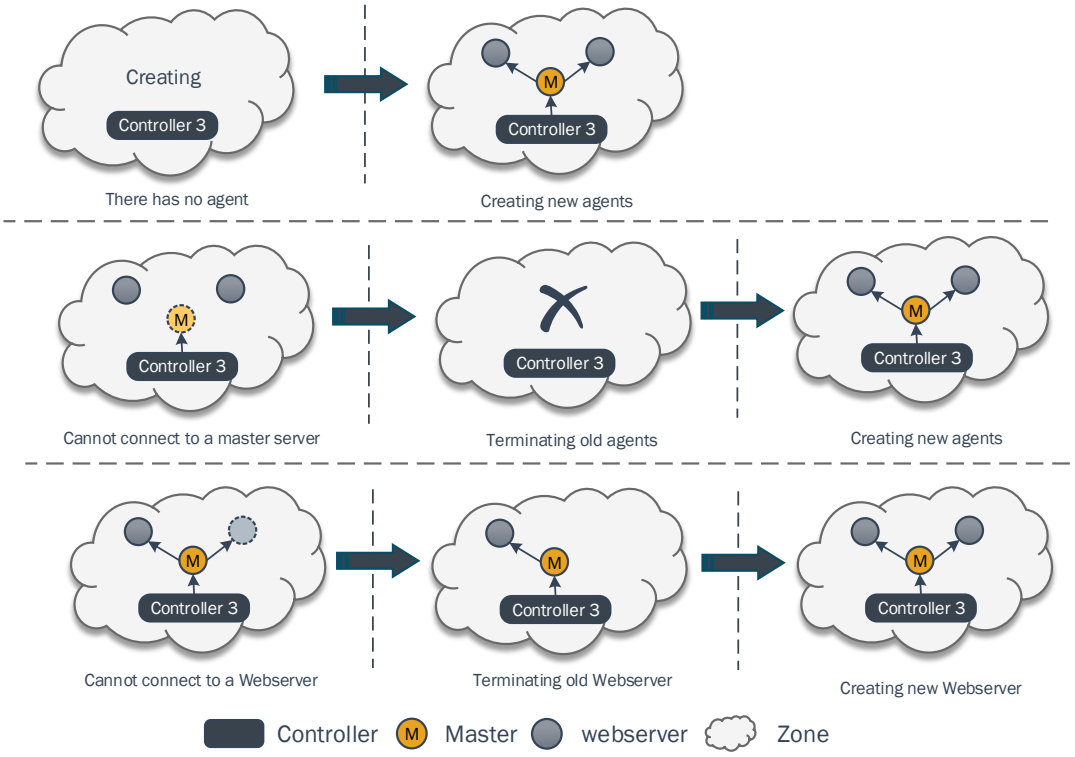


Figure 5.5: Inside Maintenance of a Zone. Recovery process in a zone is described with three scenarios about server's failure.

The controller creation mechanism has been discussed extensively in previous scenarios. But the controller in different zone is only able to create controllers in other zone. In this scenario, the possible situations within a zone will be explained. After the regeneration of a new controller, as one of its first actions, it locates and contacts with the "mother" controller. The mother controller is the controller which has created the new one. The new controller will announce itself and be registered in the hosts list of all controllers, so that it can be located too. Then, it enters into the relationships with other controllers and queries the mother controller for the policies governing it acting in its role, and obtains all the resources that it needs to make decisions about further configuration and subsequent operation.

There Has no Agent

When a new controller initializes in a new zone, this zone is considered "clean" to this controller, which means there is no agent belong to this controller should exist at this moment. This is due to a feature of configuration management software used in the agent controlling within a zone. This feature is explained following: With the security consideration, the communication from controller or master to agents is only allowed through configuration management software. In other words, there is no other means of communication between the agents is permitted. This is mainly coursed by the consideration for achieving the goal of no back doors or undocumented interfaces between agents.

The agents will regularly retrieve the configuration from master server and apply it. In order to safely communicate with the master server, the agent must have a certificate that the master server trusts. The method for the master and agent to do this, according to the configuration management software used in this project, is to run a certificate authority as part of the master and sign the certificate to specified agents. The agent will connect and request a signed certificate and use it for the further communication. It is for this reason an agent is allowed to communicate with the only master. Thus, with no other method to communicate with the agent and to change its certificate, the new master will be not able to control the agent that belongs to another master. Every new controller has to create their agents after its regeneration.

In order to ensuring the "clean" status, before the regeneration of a new controller, maintenance process will first search whole new zone for the agents which claimed to belong to a previous controller and terminate them. In fact, this work should already be done by the phase of termination of all instances in a zone as mentioned in previous scenario. Before the maintenance process start to create a new controller, it will try to terminate the old controller as well as all its agents first. The extra searching process here in the new zone is a double check in order to make sure new controller will not confusing with agents in its zone.

As stated in design chapter 4.4, the controller is responsible of creating agents involves a master server and several webservers (Two webservers have been displayed in the figure 5.5 as an example). According to the mechanism, the master server will be created and configuring ahead of webservers. Detailed configuring process will be explained in chapter 5.2.2.

Cannot Connect to a Master Server

In this scenario, a malfunctioned master server has been detected, represents the events such as a system failure occurred in partial data center that coursed some instances in that available zone have stopped working or a system corruption occurred on the master server. After the local system status checking process, the maintenance process will be invoked once the controller has realized the missing of master server or failed function on it. The controller needs to terminate the master server according to instance information record while tries to create a new one. As indicated above, since the configuration management software has been installed on the master server, all agents belong to this master need to be terminated and recreated along with the regeneration of the master. As illustrated in the figure, the controller has terminated the master and two webservers during the maintenance process, and then three new servers have been created. As each one initializes, they start to retrieve configuring instructions from their master server.

Cannot Connect to a webserver

This is the simplest scenario, that a regeneration of a malfunctioned webserver will solve it. The abnormal behavior of a webserver could be detected by analyzing the agent reports obtained from master server regularly, or local system status check. The creation of webserver is relatively straightforward, since it is the most basic component in the hydra system. However, it must be noted that, because of the concept of agent certificate, if an agent has be recreated, it will be treated as a new host and thus, it is not be allowed to grab a new certificate with the previous name. It is for this reason that the controller is required to track the history of agent names used with same master server.

5.1.2 Handling Data Storage in Hydra

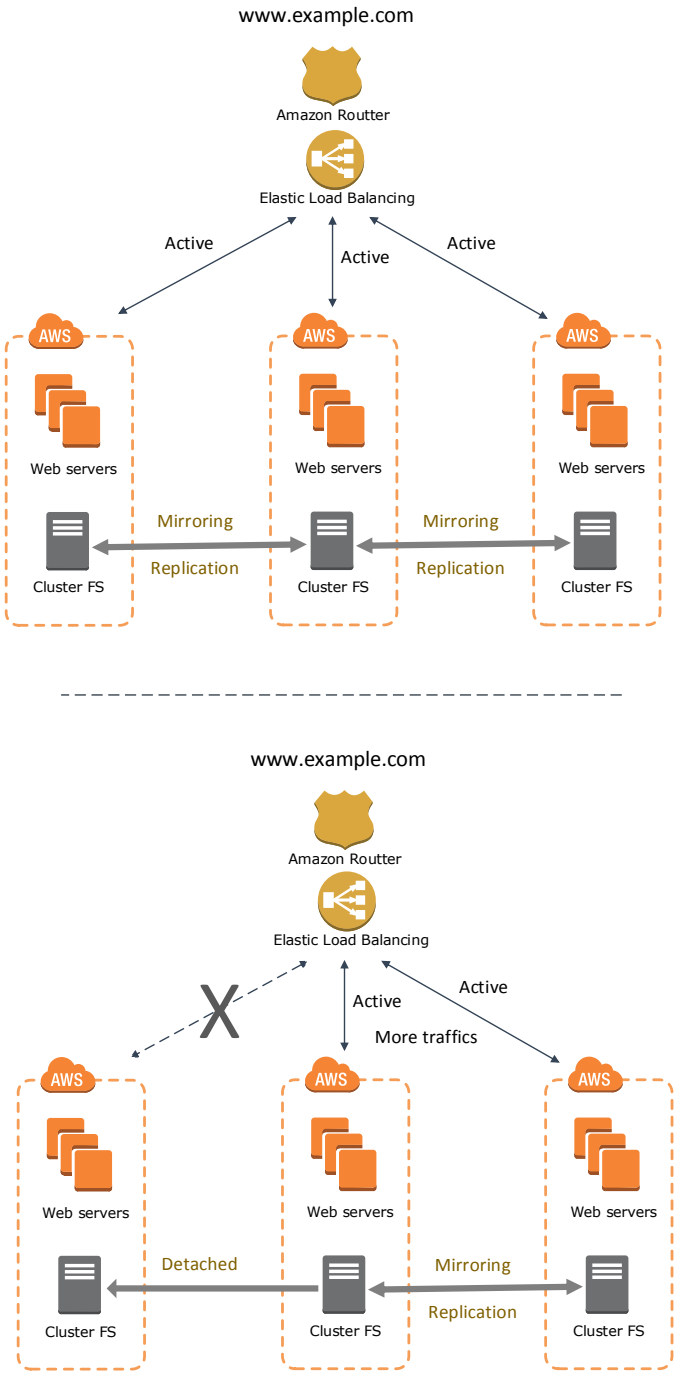


Figure 5.6: Cluster file System

Recovery Process of the Cluster File System

The description in previous chapter glosses over some potentially complex issues of hydra recovery process. This section explains how hydra system ensures the highly data availability during the recovery process. Managing data is a real and practical concern for corporate between nodes of hydra and provides many challenges for autonomic maintenance in hydra.

The figure 5.6 illustrates a web service "www.example.com" which is hosted on AWS by a hydra system which spans three availability zones. With the given DNS name, all requests sent to this website will be routed first to an Amazon module called Elastic Load Balancing [45]. This module is capable of distributing and balance incoming requests to multiple Amazon EC2 instances. In addition, another feature of this module, which will be used in this design, is that it can detect whether an Amazon EC2 instance is healthy and routes traffic to only those healthy ones. As showing in the figure, the traffics are re-routed to three sites of hydra across multiple availability zones.

In the hydra system design, every site has a storage server which is hosting cluster file system and responsible to synchronize service database. Whenever any change on one storage server will propagate to the rest of the cluster.

From this point, let us assume that one of the sites has failed. The second figure shows the recovery phase, in which one of other controllers has notified by this failure, and then it initializes the recovery process. The Elastic Load Balancing has also detected unhealthy instances at this point and automatically reroutes traffic to healthy instances in other two zones.

As introduced in previous chapter, short after the start of recovery process, the failed site is removed. Its data storage server will be detached from the cluster file system too. A new site will be created in another zone while the controller of it announces itself to other controllers. The storage server of the new site will join to the cluster file system and retrieve a copy of current cluster data from cluster file system. Finally instances in the new site register to Elastic Load Balancing and the new site goes online.

In this manner, the website is always running as long as there is one site in a zone alive. The recovery process is hidden from users and data loss is limited to the data after the last synchronization of the sites that struck by disaster or failure.

5.1.3 Configuration Management

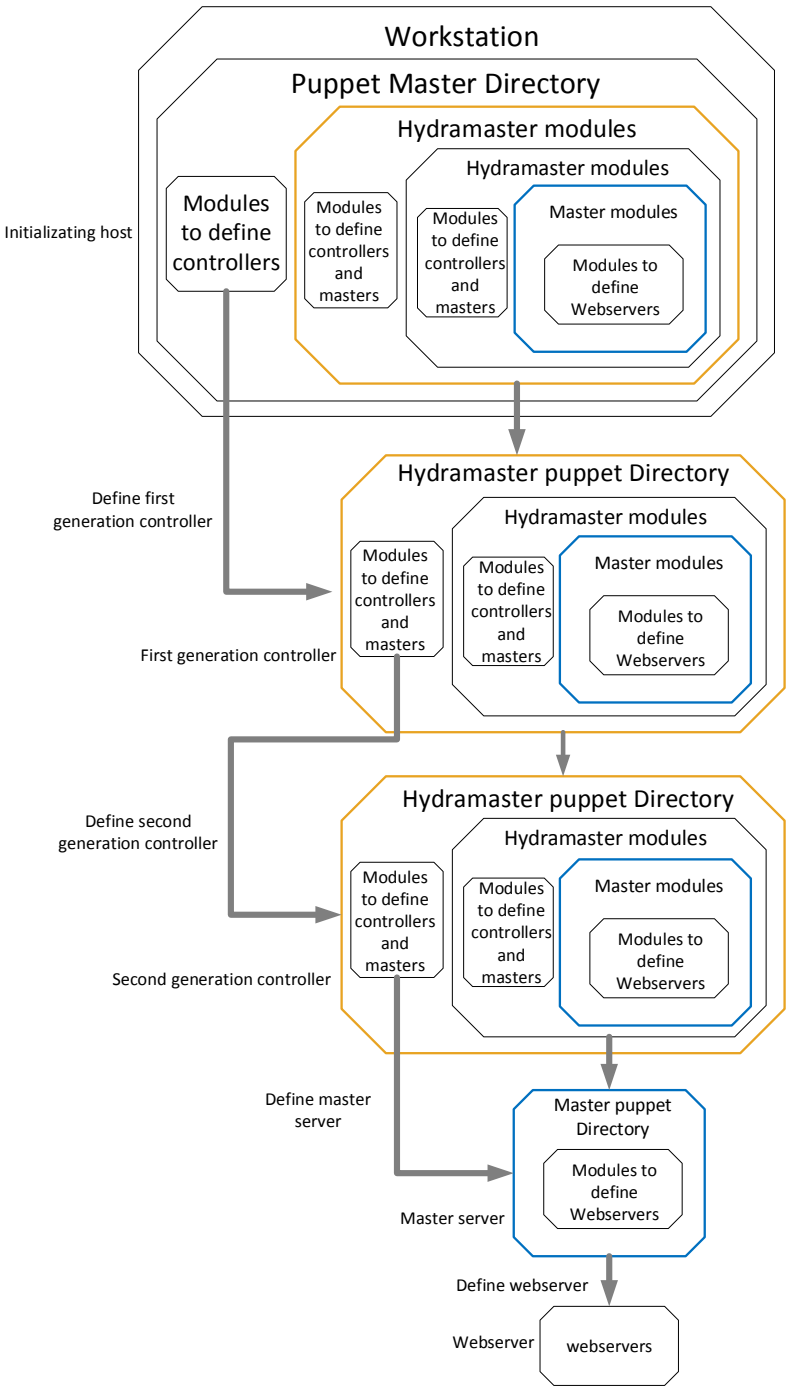


Figure 5.7: Puppet Directory

Puppet Deployment

Puppet has been chosen as the configuration management software in this design, because it is a well know open source configuration management tool which has integrated with a variety of AWS services. As indicated earlier, puppet will be in charge of configuring new hosts of hydra in an automatic manner.

In the design, puppet should behavior as both an agent and a master on the controller. As an agent, it retrieves the configuration files from master controller and applies it. As a master, it acts as a mother controller and sends out the configuration instructions to new controllers. However, a difficulty has arisen from this design. With default setup of puppet, the master is already an agent to itself. That means the master cannot to be controlled by other masters. After investigation of its configuration, it has been found the reason for this is the puppet master is using same configuration and certification directory as the puppet agent component. It signed its own agent certificate by default. Hence, with this certificate, this agent can only to communicate with master on itself. Then, how can we enable the communication with other master? One way to solve the challenge is to generate a new configuration file of puppet master manually and create a new puppet master directory in other location. However, this strategy increases the complexity of whole puppet deployment. In order to manipulate with the puppet master directory on puppet agent, an additional module is needed to copy the entire puppet directory from master to agent.

Here follows a description of the implementation related to puppet modules.

The first step of implementation is to create a controller from a local host named workstation. This controller, which is regarded as a first generation controller of hydra, is a "seed" of entire hydra system. It is capable of creating other controllers as well as servers. To illustrate the role of puppet in the creating process, the figure 5.7 in previous page gives us an overview of puppet modules on different controllers and servers of hydra. In the top of the figure, is the structure of puppet modules in the local host workstation. As illustrated in the figure, the puppet master directory of the workstation includes two parts. On the left are puppet modules which responsible for basic configurations of the first generation controller. On the right, are modules particular for creating the new puppet master directory on the first generation controller. These modules are named hydramaster modules. In order to differentiate from original puppet directory, the new puppet directory created by hydramaster modules is named hydramster puppet directory. It must be noted that, since the hydramster puppet directory is the working directory of pup-

pet master daemon on the first generation controller, in order to have same ability such as creating new controllers as workstation does. The hydramaster modules must include same modules as on the workstation. For this reason, a recursive copy of hydramaster modules has been put under hydramaster modules' directory as showing in the figure.

Blow the diagram of workstation is the diagram of puppet master directory on the first generation controller, as mentioned above, it has been named as hydramster puppet directory. Modules in this directory are copied by hydramaster modules from workstation. They have similar functions just as modules on the workstation. To better illustrate, in the hydramster puppet directory, on the left are modules required to define configurations on second generation controller, and on the right are modules to create hydramster puppet directory on second generation controller.

Apart from the configuration management of controllers, the hydra master modules are also required to manage master server. The diagram in the bottom of the figure illustrates how a controller defines a master server. In the meanwhile, since the master server is still a puppet master for webserver, the master modules imitate the previous way of creating hydramster puppet directory and creates master puppet directory on a master server. Finally, the puppet master daemon running on master server is able to define behaviors on webservers.

5.2 The Life of the Hydra

This section describes in detail of the main stages of hydra's life.

5.2.1 Stage 1: Initialization Process – Born

The journey of hydra's life starts from a single controller born on the cloud.

The deployment of the first controller is done by a local host called workstation. It must be note that, before the birth of new hydra project, it will kill previous one which has same name as it. So it is important to make sure the name is unique for every new hydra before the creating. To simplify the initialization process, there will be three controllers deployed in three separate zones simultaneously. They are called as controller 1, controller 2 and controller 3 in the following chapters. The main tool to deploy the controllers is MLN. MLN (Manage Large Networks) is a tool to build, configure and manage large groups of virtual machines based on Xen, Amazon EC2 and VMware Server[46].

In order to help automate the initializing process of hydra, there are various requirements, one of which is to preconfigure every new instance before deployed to the cloud. Then, after first boot, new instance hands control to a management tool to determine the instance type or role to ensure that the correct software and configuration is deployed. The overall goal is to have every instance end up in correct state as predefined as automatically as possible. The procedures mentioned above are performed by MLN and puppet. The benefit of using MLN is that it is a convenient tool to preconfigure instance before deployed on the Amazon cloud.

Due to the mechanism in this design, every instance is identified by the hostname. Thus, the first step of new instance deployment is to assign host name and project name to each instance. After the creation, the instance will install the puppet agent and informed with the address of puppet master.

Here is an example of appointed tasks on controller.

MLN configuration file: globalhydra.mln

```
1  ec2 {
2    type m1.small
3    user_file {
4    echo $hostname.$project > /etc/hostname
5    hostname $hostname.$project
6    apt-get update
7    apt-get -y install puppet
8    echo "127.0.0.1 $hostname" >>  /etc/hosts
9    echo "84.215.199.143  workstation" >>  /etc/hosts
10   puppet agent --verbose --no-daemonize -o -w 60 --server=workstation
11   }
12   key ke-hydra
13 }
```

The commands list above are defined the EC2 instance image type to m1.small and then signed the host and project name to it. Next, it scheduled the installation of puppet agent and execution of puppet agent command in order to communicate with puppet master after the installation.

After the Initialization process described above, the first generation controllers are running on the cloud and start to obtain all the resources from puppet master workstation for further configuration and subsequent operations.

5.2.2 Stage 2: Development – Growth

Prepare

In the stage 2, preliminary configured controllers start to locate each other and prepare for implementation of cluster file system. Glusterfs has been chosen to perform the cluster file system in hydra. Glusterfs is a well know open source platform for cluster cloud storage. It is widely used and easy to administer. After the successfully initialization of cluster storage directory, common configure files of hydra are copied to this directory. After the implementation of cluster directory, the last step for the preparation phrase is to copy maintenance scripts into cluster directory and registers them as cron job. The running state of cron job will be ensured and reported by puppet.

Maintenance in a zone

After the preparation phrase has finished, the maintenance process will start to run at regular intervals as scheduled. This process is designed to monitor and control

the functioning of master server and webservers. If the monitored servers become unresponsive or are missing, the monitoring process will take appropriate control actions such as triggering the termination and re-generation process to fix the problem. This is a part of self-healing procedure inside a zone. The procedure mentioned is handled by a script named mastercontrolling.pl. Scripts in this prototype are written by perl or bash. The programming language is not important as long as they meet the requirement of mechanism described in this prototype.

To simplify the maintenance process in the script, nonresponsive or malfunctioned instances of hydra will demonstrate as instances that have been shutdown or not exist. The main process of mastercontrolling.pl first calls a module named Test_Master. This module checks if the master server is running and return 1 if so. Else, if the master server has just been created and hence in "pending" status, it waits for a while and then repeats the checking process for at most 6 times. Or, it returns 1 to main process after the master server is ready.

When the main process received a positive response from Test_Master, it will further check the status of all webservers (or all puppet agents of the master server) by using a module named Test_Webservers. This module collects the names of missing or shutdown webservers, and then hand over the list to a Rebuild module for the creation process. Other than checking the status of webservers, the Test_Webservers module is also responsible of termination of extra webservers. The extra webservers could be servers which failed to terminate in previous process or have accidentally created by two maintenance process that run in parallel.

Otherwise, if the master server is not running or there are more than one master server exist, which means previous maintenance process may have failed, then this module will return 0 back to the main process and invoke a module named Terminate_All_Agents. This module is responsible to terminate master server and all agents (webservers) of master server. This function is needed as a requirement of certificate mechanism of puppet, this mechanism has been described thoroughly in chapter 5.1. Once the operation of Terminate_All_Agents module completed, main process will invoke a Rebuild module to create new master server and webservers.

The main process of mastercontrolling.pl shows briefly in figure 5.8 as follows,

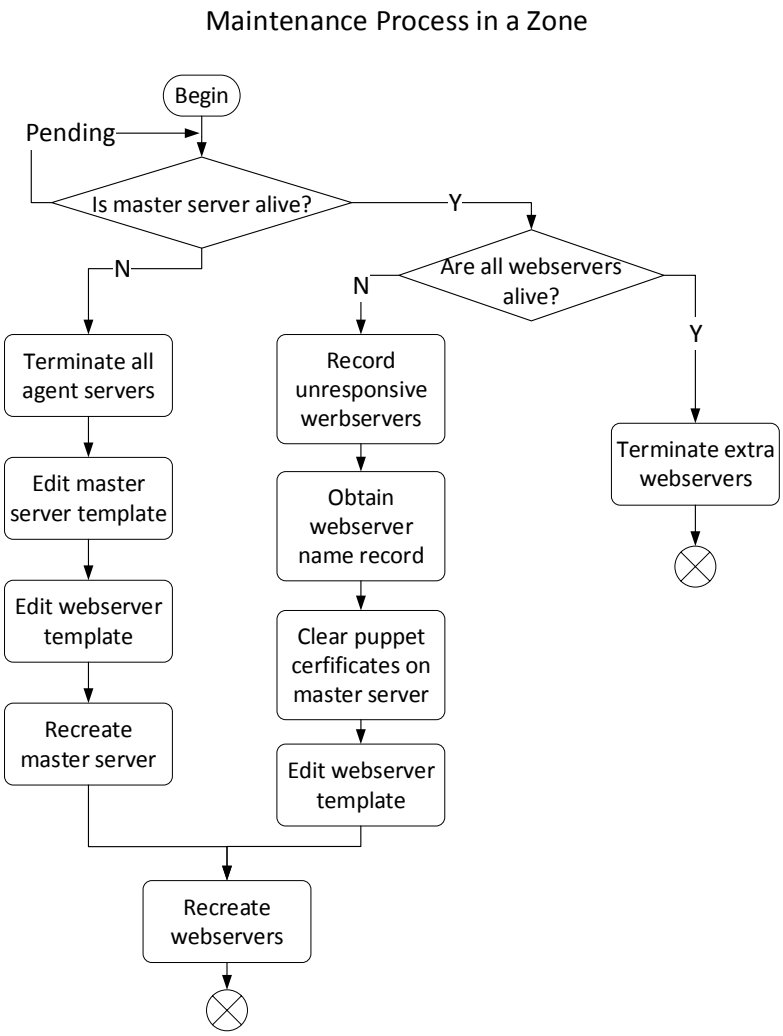


Figure 5.8: The Process of Maintenance in a Zone

After first execution of maintenance process, new instances have been generated within a zone, the hydra system has then been established. Under the monitoring of controllers, master servers and webservers are starting to serve for the Service provide by the hydra.

5.2.3 Stage 2: Maintenance – Hurt and Healing

Status checking of entire hydra is scheduled at regular intervals. This regular checking ensures the timely detecting and healing from hydra system failure.

Maintenance between zones

The regular maintenance process between zones is provided by a script named maintenance.pl. It is an autonomic manager in the hydra system. Ideally, it will have some models that map potential situations and hence chose corresponding actions according to probable outcomes. Then, when problems occur, it will to do appropriate operations. The process involves three main parts: monitor, analyze and choose right action, execute. This is the process that takes care of hydra self-healing between zones.

When controllers start the cross-zone maintenance, the execution process depends on a local knowledge of who they are, as described in section 5.1.1, the controller is only responsible for others within same hydra project and only the one has highest rank continues to take action of instance termination or creation. Other controllers with lower rank skip execute part and only report for the system failure.

The cross-zone maintenance process proceeds as follows,

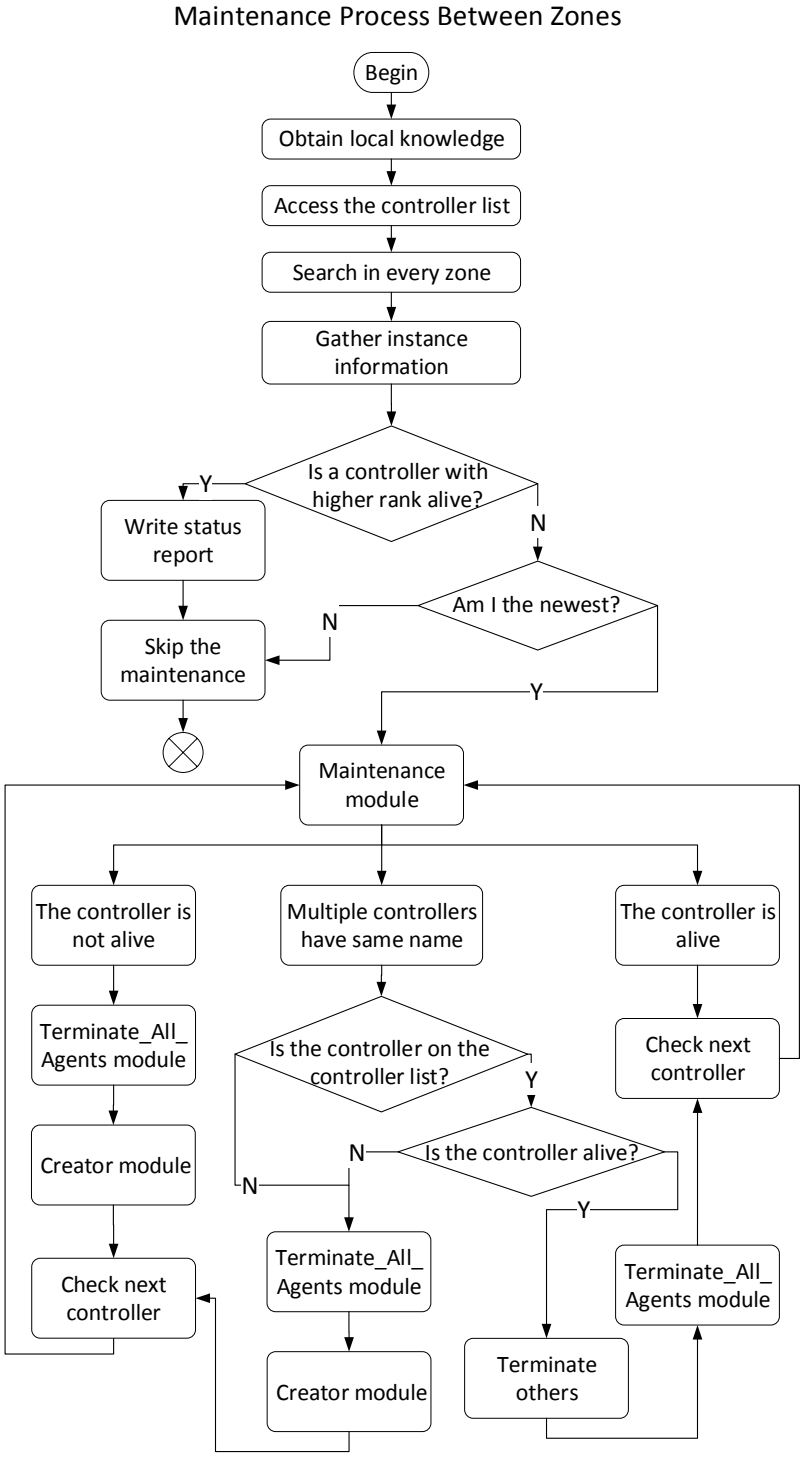


Figure 5.9: The Process of Maintenance between Zones

Monitor Part

When each controller begins to maintenance, it reads its hostname file in order to obtain the project's name and its rank. After it finished, it continues to query the instance database on the sharing cluster directory for the IP addresses of other controllers. Based on this list, any controller in the same project however not on the list will be terminated later. This is a self-protection to protect the system from impacts caused by fake or extra controllers.

Once previous work has been finished, a module named Instance_Information() executes. This module reads the list of available zones from Amazon API and then searches in every zone in order to gather information of every hydra component in entire Amazon cloud. After execution of this module, the monitor part is done.

Analyze Part

The first module executes is Check_rank, the controller compares its rank with other live controllers which on the controllers list mentioned in the monitor part above. It continues the execution only if it has the highest rank and it is the newest. The extra controller which has same rank/name with this one but created earlier will skip the maintenance, since the hydra assumes that the newer controller should be keep as it has less error as well as up to date configurations. If it really does, the Maintenance module will be triggered. This module analyzes the status of controllers for following four types:

- The controller is not alive:

  In this case, the module sends the controller's name to Terminate_All_Agents module. This sub module searches for all existed agent servers of this controller and delivers their names to Terminate module to terminate them all. This is because of the puppet certificate feature as we mentioned in previous.

- The controller is alive:

  In this case, if the controller is alive, the IP address of it will be compared with the one in the controller list to make sure it is the correct one. Otherwise, it will be send to Terminate module.

- Multiple controllers have same name:

In this case, only the controller is alive and matches the IP address in controller list will be keep. Otherwise, they will be sent to Terminate module.

After previous process, the list of missing controllers is then been delivered to Creator_queue and wait for the re-generation.

Execute Part

As part of re-generation process, the first step is performed by module Used_Regions and Sort_Region. In order to make sure the controllers work for the same project are geographically distributed. Used_Regions marks the used-rate of every zone while the Sort_Region finds the zone which has less used and near users that the site serves to.

Next, before re-generation, in order to match the requirements of new controller, MLN file is changed according to selected zone which retrieved from Sort_Region, and other profiles. After re-generation, Glusterfs_Rebuild is invoked to take care of changes in cluster file system.

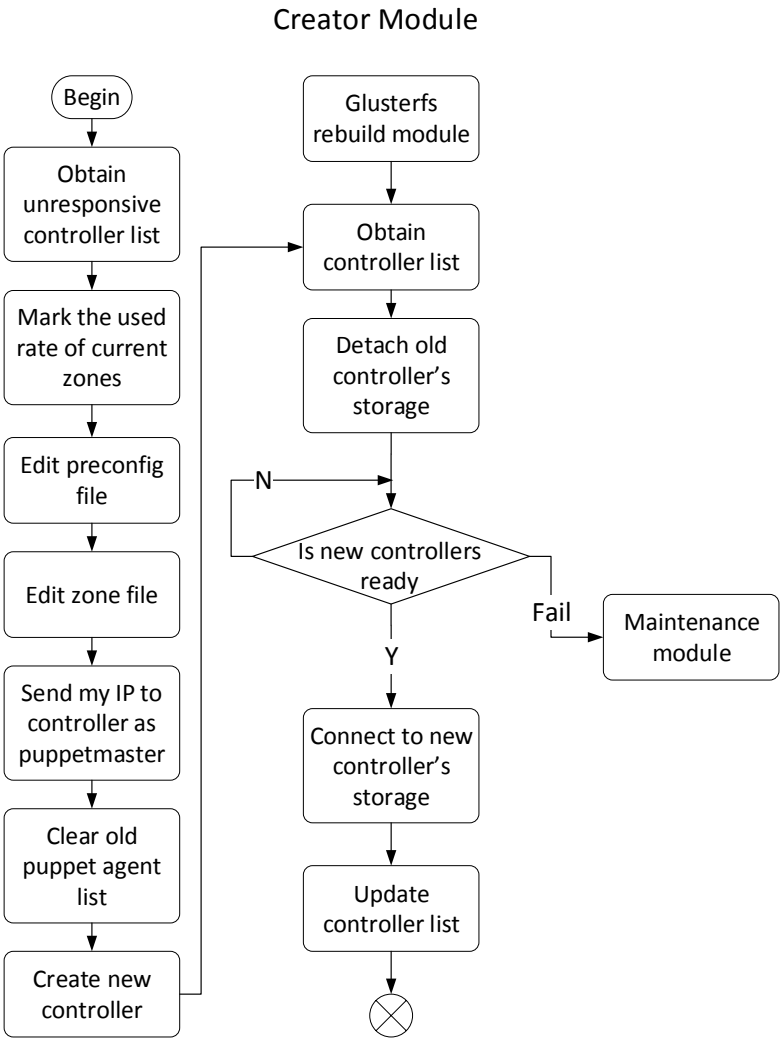


Figure 5.10: The Process of Glusterfs Rebuild

As displayed in figure 5.10, the Glusterfs_Rebuild module is a part of Creator module. After execution of create new controller command, Glusterfs_Rebuild is triggered. It queries the record of controller history to confirm current status of cluster file system. Any controller that has been replaced in previous module will be detached in this stage. Then, this module repeats to test status of new controllers until they are ready for connecting with the cluster file system. If any new controller is failed and hence becomes unresponsive during the generation, the module will hand the control back to the main process and restart maintenance of entire zone. If cluster file system has been created in the new controller successfully, the

module will end the maintenance process after updates the controller list.

The maintenance process is considered finished by the rebalance of cluster volume and controllers information updated in instance database on the sharing cluster directory.

Chapter 6

Analyze

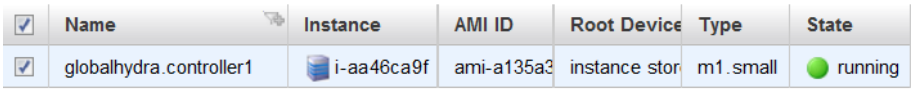
To demonstrate the power of hydra system and prove the functionality stated in the problem statement, an example of hydra project– globalhydra is running on the Amazon cloud. During the normal operation, it will encounter several system problems relevant to disaster or failure as scenarios designed in previous chapter.

6.1 The Prototype

The prototype system runs on ec2 small instances since the hydra is capable of working properly with lowest system requirement. The Service provided by the globalhydra is a web service, which provides a realistic simulation of services provided in real world. This web service is also used to show the status report of hydra system. This report will be updated every 5 minutes. Due to the time frame and consideration of expenses in the demonstration, the public DNS record of this website and Elastic Load Balancing module have not been implemented. However, the website with same report is shown by every webserver in hydra.

According to the designed model, hydra system performs system status checking regularly in order to detect system problem and to recover in time. In the prototype, the system status checking interval has been decided based on the average time required by maintenance. However, this interval could be adjusted according to system performance and recovery speed of different system.

As described in chapter 5.2, the workstation handles the initialization of hydra system. Three hydra controllers are deployed on the different regions of Amazon cloud simultaneously by MLN tool. Here is a picture of initializing of globalhydra.controller1.

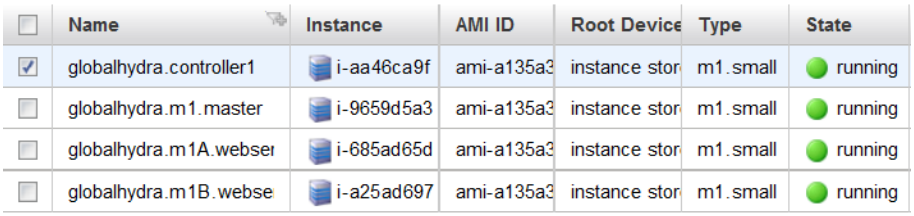


Name	Instance	AMI ID	Root Device	Type	State
globalhydra.controller1	i-aa46ca9f	ami-a135a3	instance stor	m1.small	running

Figure 6.1: The globalhydra controller 1 is initializing on the Amazon cloud in region US West (Oregon).

During the initialization process, controllers retrieve configuration files from workstation and prepare for the first execution of maintenance process. Since all controllers are present, the controllers start to create their agents (master server and webservers) in the zone they live in.

Here is a picture shows the state of zone 1 (where controller 1 lives) after first execution of maintenance.



Name	Instance	AMI ID	Root Device	Type	State
globalhydra.controller1	i-aa46ca9f	ami-a135a3	instance stor	m1.small	running
globalhydra.m1.master	i-9659d5a3	ami-a135a3	instance stor	m1.small	running
globalhydra.m1A.webser	i-685ad65d	ami-a135a3	instance stor	m1.small	running
globalhydra.m1B.webse	i-a25ad697	ami-a135a3	instance stor	m1.small	running

Figure 6.2: After a while, the master server and webservers have been generated in the zone same with their controller.

After the hydra initialization has completed, the continuous reports of hydra system are showing on every webserver. In order to reduce processing loads on the controllers, the status report is generated by master servers and then pushed to every webservers.

Status Report Example of Controller 1 and Controller 2

```
1  Controller1 is running on region US West (Oregon)
2  Launch time: 2013-05-19 20:04:06
3  IP address: 54.214.52.136
4  ****************************
5  Master1 is running
6  Launch time: 2013-05-19 20:37:32
7  IP address: 54.214.155.235
8  WebserverA is running
9  Launch time: 2013-05-19 20:39:46
10 IP address: 54.214.156.110
11 WebserverB is running
```

```
Launch time: 2013-05-19 20:40:21
IP address: 54.214.87.99
****************************
Controller2 is running on region EU West (Ireland)
Launch time: 2013-05-19 20:04:20
IP address: 54.216.98.50
****************************
Master2 is running
Launch time: 2013-05-19 20:19:49
IP address: 54.216.83.241
WebserverA is running
Launch time: 2013-05-19 20:22:14
IP address: 46.51.161.14
WebserverB is running
Launch time: 2013-05-19 20:22:59
IP address: 46.51.134.38
```

According to the report above, the controller 1 has launched in 20:04:06 and the webserverB, which is the last server be created, has launched in 20:40:21. That shows the initialization process in zone 1 (controller 1 lives) was about 40 minutes. However, the same process in zone 2 (controller 2 lives) was only about 15 minutes. Based on experiences of performing same process on different zones several times, it was found that the performances of instances are slightly different according to location of the instance. It may caused by local network latency or physical infrastructure of cloud. This difference has affected the implementation of cluster file system in hydra and the effect will be discussed in next chapter.

6.2 Implementation of Scenarios

Three common scenarios have been tested with the prototype.

6.2.1 Regeneration of Controller

First, controller 1 has been shutdown. According to the model, this situation is regarded as an unstable controller or a disaster in whole zone. All servers in this zone will be terminated while the new controller 1 generated in a new zone.

Name	Instance	AMI ID	Root Device	Type	State
globalhydra.controller1	i-aa46ca9f	ami-a135a3	instance stor	m1.small	shutting-
globalhydra.m1.master	i-9659d5a3	ami-a135a3	instance stor	m1.small	running
globalhydra.m1A.webser	i-685ad65d	ami-a135a3	instance stor	m1.small	running
globalhydra.m1B.webse	i-a25ad697	ami-a135a3	instance stor	m1.small	running

Figure 6.3: The controller 1 has been shut down as an example of a disaster has struck zone 1.

After a while, the error message has been found on the report which hosting on webservers.

Controller 1 is unresponsive

```
1 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
2 Controller1 is unresponsive
3 ****************************
4 Controller2 is running on region EU West (Ireland)
5 Launch time: 2013-05-19 20:04:20
6 IP address: 54.216.98.50
```

After next maintenance process, controller 2 has noticed this error and generates new controller 1 in another zone while all servers on original zone 1 have been terminated.

Name	Instance	AMI ID	Root Device	Type	State
globalhydra.controller1	i-aa46ca9f	ami-a135a3	instance stor	m1.small	terminated
globalhydra.m1.master	i-9659d5a3	ami-a135a3	instance stor	m1.small	terminated
globalhydra.m1A.webser	i-685ad65d	ami-a135a3	instance stor	m1.small	terminated
globalhydra.m1B.webse	i-a25ad697	ami-a135a3	instance stor	m1.small	terminated

Figure 6.4: The controller 1 has been shut down as an example of a disaster strikes zone 1.

Next, after the new controller 1 is ready, it creates all servers within its zone.

Name	Instance	AMI ID	Root Device	Type	State
globalhydra.m1A.webserverA1	i-bb9b92a7	ami-8870aa	instance stor	m1.small	running
globalhydra.controller1	i-2a505b36	ami-8870aa	instance stor	m1.small	running
globalhydra.m1B.webserverB1	i-5651554a	ami-8870aa	instance stor	m1.small	running
globalhydra.m1.master	i-0beeca17	ami-8870aa	instance stor	m1.small	running

Figure 6.5: The controller 1 has been generated in a other zone.

The following report shows a new controller 1 has been launched at 21:07:04 in a new zone Sao Paulo. After its regeneration, it noticed the master server is not present yet.

New Controller 1 Has Been Created

```
1
2  Controller1 is running on region South America (Sao Paulo)
3  Launch time: 2013-05-19 21:07:04
4  IP address: 54.232.42.86
5  ****************************
6  Master1 is unresponsive
```

After about 30 minutes, the regeneration of all servers in new zone has been completed.

All Servers in New Zone 1 Have Been Created

```
1  Controller1 is running on region South America (Sao Paulo)
2  Launch time: 2013-05-19 21:07:04
3  IP address: 54.232.42.86
4  ****************************
5  Master1 is running
6  Launch time: 2013-05-19 21:37:33
7  IP address: 54.232.78.11
8  WebserverA is running
9  Launch time: 2013-05-19 21:39:55
10 IP address: 54.232.31.229
11 WebserverB is running
12 Launch time: 2013-05-19 21:40:35
13 IP address: 54.232.56.156
```

6.2.2 Regeneration of Master Server

Second, the master server in zone 3 has been shutdown to simulate the outrage with master server.

Name	Instance	AMI ID	Root Device	Type	State
globalhydra.controller3	i-76d0f32e	ami-2fb897(	instance stor	m1.small	running
globalhydra.m3.master	i-06a2815e	ami-2fb897(	instance stor	m1.small	shutting-
globalhydra.m3A.webser	i-78a18220	ami-2fb897(	instance stor	m1.small	running
globalhydra.m3B.webse	i-18a18240	ami-2fb897(	instance stor	m1.small	running

Figure 6.6: The master server in zone 3 has been shutdown.

This outrage has been detect and the report shows information as follows,

Master 3 is Unresponsive

```
1 Controller3 is running on region US West (N. California)
2 Launch time: 2013-05-19 20:04:35
3 IP address: 54.241.236.77
4 ****************************
5 Master3 is unresponsive
```

Controller 3 triggers the maintenance process within zone 3. According to the mechanism discussed in chapter 5.2, while the regeneration of new master server, all webservers have been terminated.

Name	Instance	AMI ID	Root Device	Type	State
globalhydra.controller3	i-76d0f32e	ami-2fb897(	instance stor	m1.small	running
globalhydra.m3.master	i-06a2815e	ami-2fb897(	instance stor	m1.small	terminated
globalhydra.m3A.webser	i-78a18220	ami-2fb897(	instance stor	m1.small	terminated
globalhydra.m3B.webse	i-18a18240	ami-2fb897(	instance stor	m1.small	terminated
globalhydra.m3.master	i-b8ac8fe0	ami-2fb897(	instance stor	m1.small	running
globalhydra.m3A.webser	i-0ab39052	ami-2fb897(	instance stor	m1.small	running
globalhydra.m3B.webse	i-36b3906e	ami-2fb897(	instance stor	m1.small	running

Figure 6.7: All agent servers in zone 3 have been terminated.

When master server 3 is ready, the status report is sent as follows,

New Master 3 Has Been Created

```
1 ****************************
2 Master3 is running
3 Launch time: 2013-05-19 21:21:33
4 IP address: 54.215.86.128
5 WebserverA is unresponsive
6 WebserverB is unresponsive
```

As it shows in the report above, webservers in zone 3 are need to be created after the recreation of master server 3.

The recreation of webservers is pretty quick. According to report follows, it used only 2 minutes. The reason for this is the recreation of webservers has always been ordered along with the recreation of master sever, since original webservers have been all terminated in previous steps. And finally, the zone 3 is back to normal operation.

New Master 3 Has Been Created

```
1  Controller3 is running on region US West (N. California)
2  Launch time: 2013-05-19 20:04:35
3  IP address: 54.241.236.77
4  ****************************
5  Master3 is running
6  Launch time: 2013-05-19 21:21:33
7  IP address: 54.215.86.128
8  WebserverA is running
9  Launch time: 2013-05-19 21:23:36
10 IP address: 184.169.209.125
11 WebserverB is running
12 Launch time: 2013-05-19 21:24:14
13 IP address: 184.169.203.35
```

6.2.3 Regeneration of Webserver

Finally, webserverB1 in zone 2 has been shut down to simulate the outrage with webservers.

Name	Instance	AMI ID	Root Device	Type	State
globalhydra.controller2	i-9c9835d1	ami-f97a6b	instance stor	m1.small	running
globalhydra.m2.master	i-20913c6d	ami-f97a6b	instance stor	m1.small	running
globalhydra.m2A.webser	i-0c903d41	ami-f97a6b	instance stor	m1.small	running
globalhydra.m2B.webse	i-d6903d9b	ami-f97a6b	instance stor	m1.small	shutting-d

Figure 6.8: The webserverB1 in zone 2 has been shut down.

This outrage is detected and shown in the status report as follows,

WebserverB is Unresponsive

```
1  Controller2 is running on region EU West (Ireland)
2  Launch time: 2013-05-19 20:04:20
3  IP address: 54.216.98.50
```

```
4  ****************************
5  Master2 is running
6  Launch time: 2013-05-19 20:19:49
7  IP address: 54.216.83.241
8  WebserverA is running
9  Launch time: 2013-05-19 20:22:14
10 IP address: 46.51.161.14
11 WebserverB is unresponsive
```

The recreation of webserver is relatively straightforward, since it is the most basic component in the hydra system. However, according to mechanism mentioned in section 5.1, it is not allowed to use previous name. The new webserverB1 will hence be named webserverB2. The name change is not showing in the report since this information is only required the maintenance process.

Name	Instance	AMI ID	Root Device	Type	State
globalhydra.controller2	i-9c9835d1	ami-f97a6b	instance stor	m1.small	running
globalhydra.m2.master	i-20913c6d	ami-f97a6b	instance stor	m1.small	running
globalhydra.m2A.webserverA1	i-0c903d41	ami-f97a6b	instance stor	m1.small	running
globalhydra.m2B.webserverB1	i-d6903d9b	ami-f97a6b	instance stor	m1.small	terminated
globalhydra.m2B.webserverB2	i-a27bd6ef	ami-f97a6b	instance stor	m1.small	running

Figure 6.9: The new webserverB2 in zone 2 has been created.

After the maintenance process with zone 2 has been finished, the webserverB2, which shows in the last line of the picture above, is running and starts to host the website again.

WebserverB Has Been Recreated

```
1  Controller2 is running on region EU West (Ireland)
2  Launch time: 2013-05-19 20:04:20
3  IP address: 54.216.98.50
4  ****************************
5  Master2 is running
6  Launch time: 2013-05-19 20:19:49
7  IP address: 54.216.83.241
8  WebserverA is running
9  Launch time: 2013-05-19 20:22:14
10 IP address: 46.51.161.14
11 WebserverB is running
12 Launch time: 2013-05-19 21:19:43
13 IP address: 176.34.207.184
```

As displayed in the report, the webserverB's launch time and the IP address has

been updated according to the instance information of webserverB2.

Chapter 7

Discussion

Approaches of previous work vary from model design to prototype development. The model presented in previous chapter demonstrates thoroughly the mechanism introduced in this thesis. The prototype, as a proof of the mechanism concept, offers a basic implementation of the model. In this chapter, the model design, the whole architecture, the prototype, and the complications during the prototyping are discussed. The choices made during the prototype development are discussed, and possible modifications to the architecture and future work are suggested.

7.1 Overview of the model

This section explains some feathers of the model related to problem statement.

7.1.1 Automated

This thesis demonstrates how an automatic approach can be used to address the difficulties in a disaster recovery plan on the cloud platform. Automatic approach is used to address complexity of modern computing system, as well as lowers cost of infrastructure and administration. The automatic approach introduced in this thesis, as describe in the problem statement, covers the requirements include *auto-growth*, *auto-detecting* and *self-healing*.

Looking at the results from the functionality testing of the prototype in analyze chapter, it is evident that the prototype is able to address the feature of *auto-growth*. In the initialization phase of hydra system, there is only controllers will be deployed on the cloud (section 5.2.1). When a controller has been generated in a zone, it is in charge of creation process of its agent servers such as master server

and webserver right after its initialization (section 5.2.2). If a controller is missing, other controllers are capable of creating new controller as well (section 5.2.3). The entire creation process will complete when every components of hydra have been presented.

After the initialization phase finished, the hydra starts normal operation phase (section 5.2.3). In this phase, regular status checking of hydra system is scheduled. Maintenance process running on each controller takes care of tasks such as outrage detection and system recovery. As requirement to *auto-detection*, the detection process runs at regularly intervals automatically. Once the hydra system has been "hurt" by a disaster or a failure on any of its components, the outrage will be detected by every live controller and be reported. This automatic approach has minimizing the detection time to less than the interval of every maintenance process.

After the system outrage has been confirmed, restore process will then be triggered without a need of manual intervention. The live controller that has highest rank will take charge of restore process. Like its name hydra indicates, as long as there is a controller alive, the hydra system heals itself automatically.

7.2 Adaptable

As pointed out in background chapter, although Disaster Recovery (DR) is a desirable feature for all enterprises, adoption of disaster recovery plan remains limited. Traditional disaster recovery service normally falls in constraints such as high cost of infrastructure, since they require a primary site to provide service, but need to pay extra for a secondary site for data backup in the same time [47, 13, 48, 49]. According to the result from prototype testing (section 6.1), the hydra system has same feature of data backup. However, instead of paying for a secondary site which used when the primary site is unavailable. Each site in hydra is regarded as primary site and responsible for providing service to end users. Extra site will only be paid when needed or after the termination of old site.

7.3 Reliable

In order to achieve the feature of *Reliable* and ensure *Service* consistency, each site of hydra keeps synchronizing replication data of *Service* by using cluster file system. Meanwhile, hydra system distributes *Service* traffics to different site (sec-

tion 5.1.2) and scheduled status checking of entire hydra at regular intervals. This interval depends on the cost of running frequency of maintenance process. Normally depends on the decision of Recovery Time Objective (RTO), RTO refers to the amount of downtime that is acceptable before the system return to the normal operation state. Although in the traditional DR plan, a secondary site is keeping in a standby state, the service downtime is inevitable since it may take minutes to bring it online after failure. However, in the design of hydra, every site is regarded as primary site and handles traffics for a same *Service*. The service downtime can only be coursed by all site controllers are destroyed in the same time, or decreased overall performance during restore process of partial hydra nodes. When a site has declared a failure, the traffics will then redirect to sites that are still available. Moreover, the traffics overhead of the live sites could be avoided by using more powerful instance. This approach also ensures the *data availability*, because the data is always distributed and unlikely to become available because one individual site fails.

7.4 Technical Pitfalls and Possible Modifications

As outlined above, the mechanism presents in this thesis can be used to help automated system disaster recovery procedure and avoid service downtime. However, the implementation of the mechanism in a real world model needs further improvement. In the approach chapter, utilizing existing tools has been chosen in order to simplify the models development and make the prototype easier to be adapted by administrators that have former knowledge with those tools. However, there are unexpected technical complications arose during the process of development of the prototype. Some of the technical pitfalls encountered are described as follows.

7.4.1 Puppet

The puppet has been used in the prototype as a main approach to achieve auto configuration in an instance. In the design chapter, two layers architecture has been chosen in order to separate the structure of hydra system from the Services hosted on agent servers. However, the layered approach has increased the complexity of implementation of puppet (section 5.1.3). Original puppet configure do not support a puppet master working as an agent to another puppet master. In order to achieve this feature, a recursive multilevel puppet directory has been implemented. There could possibly have an alternative method which is simpler to perform auto configuration in instance other than puppet, but requires further investigation.

A difficulty arose from the complex implementation is handling dependencies in the configuring process. In a complex environments, it is important to avoid deadlock or circular dependencies when setup puppet rules. During auto configuration process of an instance, some processes needs to follow certain sequences. One point of failure will result in failures in all following operation. However, the multilevel puppet directory greatly complicates the process to setup proper rules for different agents.

Other than the technical limitations, the approach of asking new controller always retrieves configure files and resource from mother controller contains a potential problem (section 5.1.1). If the mother controller becomes unresponsive right after the creation, the initialization process of new controller will fail. An alternative approach would be to obtain the resource from any existing controllers. Each controller would have the capability to provide resource to others within some project.

7.4.2 Cluster File System

The purpose of a self-healing system is to provide reliability and data integrity in the face of unforeseen disaster or system failure. This approached by adding a cluster file system on geographically distributed sites of hydra system (section 5.1.2). Glusterfs was used to implement cluster file system in the hydra since it is widely used and easy to administer.

Creating a cluster file system with Glusterfs is straightforward. However, when it combines with puppet, some problems have been posed. One is the timing to mount cluster file system. In the prototype, three controllers are generating in the same time (section 5.2.1), assuming controller 1 will responsible of initializing the cluster storage, two others will need to wait the cluster storage has successfully created before they mount the cluster directory. And data will copy to cluster storage until the directory has come up. However, there is no approach to control the sequence of puppet policy implementations between different agents. A compromise method is to tell the two controllers keeping status checking of cluster storage, and mount it after a success signal. But if there is network latency or whatever other reasons delay the cluster initializing, the puppet process on the two controllers may fail because of the timeout of status checking command. Nonetheless, the timing issue has been solved by a trick of executing cluster initializing command on every controller before the mounting. As a result, the one who first executes the command will success and the others will fail, but when they start the mounting and coping

after the command they will success since the cluster storage has been initialized by the first one.

Although cluster mounting command works fine in command line window, it fails when using puppet to execute it. It was hard to find a solution since using puppet to work with Glusterfs is new. This problem has been eventually solved by adding the loopback address 127.0.0.1 to DNS of every controller, but the reason behind it was not investigated due to time constraints. In the original design, when dealing with duplicated instances and hydras, it was planned to keep only the newest instance, since the newer instance normally has less error as well as up to date configurations. However, when performing cluster file system reattach with new instance from other hydra (section 5.1.1 scenario 4), it shows that if a instance was already be a part of a cluster file system, it does not allowed to join another. This limitation has changed the mechanism instead to locate and keep instances which already in the same cluster with the controller.

Another issue is the speed of cluster file system. The cluster file system relies heavily on low latency network environment. It works fine if all nodes are in same zone. However, when separate them to different zone, the cluster suffers from a great delay, even there is no ongoing data synchronization. In the prototype, in order to demonstrate the cluster file system in hydra system, MLN files are stored in the cluster directory. The performance of MLN has greatly influenced by I/O speed of the cluster. The MLN works well when keeping a few configuration files in the cluster directory. However, if more data have been storing in it, the MLN will fail. It is surprising how the latency of network greatly affects the performance of cluster file system. However, further experiments suggest the performance bottlenecks of cluster can be improved by using more powerful instance and high speed network. But such a choice would perhaps not be justifiable in terms of costs.

The above description gives an overview of unexpected technical obstacles of tools used in this thesis. Some obstacles encountered during the prototype development have great influence to the original design. For example, if the puppet issue cannot be solved by multilevel puppet directory, the implementation of auto configuration in hydra will have to be completely changed. However, there may be a replacement or alternative method to achieve a better result, but it requires more time to investigate, which is unpractical due to time constraints in this thesis. So it would be wise to make use of different tools before starting the design.

7.5 Future Work and Suggested Improvements

7.5.1 Multi-cloud Approach

The Amazon cloud environment is amazing, but it's still not a fit for all jobs, and even if it was, it wouldn't be wise to place everything there. In order to diversify risk, while to better suit the needs of specific workload/ requirements, moving hydra to a multi-cloud environment would be a promising solution. Although each cloud provider has designed their architecture of hardware and software differently, with necessary changes to codes and using specific APIs, tools provided by a cloud platform, hydra might able to be expanded to multi-cloud and even select suitable cloud infrastructure for a particular task.

7.5.2 User Interface

In order to ensure the reliability, hydra system is designed to running on itself without relying on anyone else. This feature might lead to a difficulty of controlling behavior of hydra. It would be interesting to develop a user interface that allows an administrator to observe the current state of hydra and interact with each autonomic component in it. The user interface could be a web application consisting of a number of reports about the current status of servers or performance values for each application environment. It may also possible to utilize IBM's Integrated Solutions Console [21], which is an interface framework that is itself built on WebSphere Portal technology. It communicates with the autonomic elements in the system through their defined Web-Services interfaces.

7.5.3 Optimization of Cluster File System

An additional work should be relevant to cluster file system in hydra. Synchronous replication stays at the foundation of hydra, however, as mentioned in chapter 7.4.2 the performance of Glusterfs is barely satisfactory. Available techniques today may still not completely meet the needs of the mechanism in the thesis. However, this issue will possibly be solved by the help of some novel techniques such as pipelined synchronous replication [44] which optimizes the replication process for virtual machines and synchronizes replication throughout the nodes that make up a distributed service. Or by architecting data distribution schemes: combinations of striping, replication, and erasure-coding are used to store data on multiple sites [50] etc..

7.5.4 Encapsulation

The purpose of using layered structure in original design is to encapsulate entire hydra system to a "black-box". With no need of interference with the hydra structure, customer services can be easily integrated with hydra by interface without altering any other part of application. This will be a perfect fit for the arbitrary nature of VM-based cloud hosting. However, the prototype developed in this thesis does not fully meet the requirement of this purpose. This will need to be address in future work.

Chapter 8

Conclusion

In this thesis, a disaster recovery solution which aims to achieve an automated, adaptable and reliable recovery process on a cloud based system has been proposed.

First, a novel mechanism has been proposed to automate the disaster recovery process. Then, based on the mechanism a model has been designed and thoroughly described. Finally, the feasibility and efficiency of the model is verified by a prototype. The experiment results illustrate the potential for automating disaster recovery using cloud platforms.

In the model, each controller is responsible for its own internal autonomic behavior, namely, managing the data resources in local site, managing its own internal operations including monitoring local site status, managing agent servers. Each controller is also responsible for forming and managing relationships between controllers, to accomplish its goals, that is, the external autonomic behavior that enables the system as a whole to be self-managing, including auto-growth, auto-detecting and self-healing.

Although this work is still in a preliminary stage and some practical issues that will need to be addressed in future work, it opened a valuable exploration to find a powerful and flexible way to allow system recovery from disaster by itself. It greatly reduces the amount of time and resources to monitor system. If a disaster occurs, a replacement will be automatically launched while the original be removed.

Bibliography

[1] H. S. Gunawi, T. Do, J. M. Hellerstein, I. Stoica, D. Borthakur, and J. Robbins, "Failure as a service (faas): A cloud service for large-scale, online failure drills," *University of California, Berkeley, Berkeley*, vol. 3, 2011.

[2] digi.no, "Flom slo ut datasenter," http://m.digi.no/916961/flom-slo-ut-datasenter?utm_source=feedly, May 2013.

[3] M. W. Thomas, "Rackspace resolves email outage following possible denial of service attack," http://www.bizjournals.com/sanantonio/news/2013/01/14/rackspace-resolves-email.html, Jan 2013.

[4] D. Harris, "Amazon blames human error for xmas eve outage; netflix vows better resiliency," http://gigaom.com/amazon-blames-human-error-for-xmas-eve-outage-netflix/, December 2012.

[5] D. Goldman, "Amazon explains its cloud disaster," http://money.cnn.com/2011/04/29/technology/amazon_apology/index.htm, April 2011.

[6] Amazon, "Summary of the december 24, 2012 amazon elb service event in the us-east region," https://aws.amazon.com/message/680587/, Dec 2012.

[7] S. Ludwig, "Amazon cloud outage takes down netflix, instagram, pinterest," http://venturebeat.com/2012/06/29/amazon-outage-netflix-instagram-pinterest/, June 2012.

[8] O. Malik, "Parts of amazon web services suffer an outage," http://gigaom.com/2012/06/14/did-amazons-web-services-go-down/, June 2012.

[9] J. Scott, "Amazon and microsoft cloud services hit by lightning strike," http://www.cloudpro.co.uk/cloud-essentials/1453/amazon-and-microsoft-cloud-services-hit-lightning-strike, August 2011.

[10] B. Butler, "Amazon outage one year later: Are we safer?" http://www.networkworld.com/news/2012/042712-amazon-outage-258735.html, April 2012.

[11] H. Blodget, "Amazon's cloud crash disaster permanently destroyed many customers' data," http://www.businessinsider.com/amazon-lost-data-2011-4, April 2011.

[12] V. Sarathy, P. Narayan, and R. Mikkilineni, "Next generation cloud computing architecture: Enabling real-time dynamism for shared distributed physical infrastructure," in *Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on*. IEEE, 2010, pp. 48–53.

[13] C. S. U. Omar H. Alhazmi, Taibah University Yashwant K. Malaiya, "Evaluating disaster recovery plans using the cloud," *Software Reliability Engineering*, 2012.

[14] R. Murch, *Autonomic computing*. IBM Press, 2004.

[15] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[16] S. S. Laster and A. O. Olatunji, "Autonomic computing: Towards a self-healing system," *Proceedings of the Spring*, pp. 62–78, 2007.

[17] J. Kephart, "Autonomic computing: the first decade," in *Proceedings of the 8th ACM international conference on Autonomic computing-ICAC*, vol. 11, 2011, p. 1.

[18] J. O. Kephart, "Research challenges of autonomic computing," in *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*. IEEE, 2005, pp. 15–22.

[19] A. Moura, J. Sauvé, and C. Bartolini, "Research challenges of business-driven it management," in *Business-Driven IT Management, 2007. BDIM'07. 2nd IEEE/IFIP International Workshop on*. IEEE, 2007, pp. 19–28.

[20] A. G. Ganek and T. A. Corbi, "The dawning of the autonomic computing era," *IBM systems Journal*, vol. 42, no. 1, pp. 5–18, 2003.

[21] IBM, "Ibm. security solutions," http://www-03.ibm.com/security/cisco, April 2013.

[22] S. Hariri, "Autonomic computing: research challenges and opportunities," in *Pervasive Services, 2004. ICPS 2004. Proceedings. The IEEE/ACS International Conference on*. IEEE, 2004, p. 7.

[23] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[24] M. D. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali, "Cloud computing: Distributed internet computing for it and scientific research," *Internet Computing, IEEE*, vol. 13, no. 5, pp. 10–13, 2009.

[25] H. Brian, T. Brunschwiler, H. Dill, H. Christ, B. Falsafi, M. Fischer, S. G. Grivas, C. Giovanoli, R. E. Gisi, R. Gutmann *et al.*, "Cloud computing," *Communications of the ACM*, vol. 51, no. 7, pp. 9–11, 2008.

[26] A. Iosup, R. Prodan, and D. Epema, "Iaas cloud benchmarking: Approaches, challenges, and experience," in *proceedings of the ACM/IEEE Conference on High Performance Networking and Computing (SC), MTAGS*. IEEE/ACM, pp. 1–8.

[27] Wikipedia, "Cloud computing," http://en.wikipedia.org/wiki/Cloud_computing, April 2013.

[28] N. Leavitt, "Is cloud computing really ready for prime time," *Growth*, vol. 27, no. 5, 2009.

[29] A. AWS, "About aws," http://aws.amazon.com/about-aws/, April 2013.

[30] A. EC2, "Amazon elastic compute cloud (amazon ec2)," http://aws.amazon.com/ec2/, April 2013.

[31] A. W. Service, "Disaster recovery using amazon web services," http://www.slideshare.net/AmazonWebServices/using-aws-for-disaster-recovery-webinar, April 2013.

[32] A. D. Board, "Amazon web services dash board," http://status.aws.amazon.com/, April 2013.

[33] D. Mendoza, "Amazon outage takes down reddit, foursquare, others," http://edition.cnn.com/2012/10/22/tech/web/reddit-goes-down, October 2012.

[34] O. H. Alhazmi and Y. K. Malaiya, "Assessing disaster recovery alternatives: On-site, colocation or cloud," in *Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on*. IEEE, 2012, pp. 19–20.

[35] G. Results, "Symantec 2010 disaster recovery study," November 2010.

[36] C. Preimesberger, "Survey indicates half of smbs have no disaster recovery plan," http://www.eweek.com/c/a/Data-Storage/Survey-Indicates-Half-of-SMBs-Have-No-Disaster-Recovery-Plan-687524/, Sep 2009.

[37] T. Wood, E. Cecchet, K. Ramakrishnan, P. Shenoy, J. Van der Merwe, and A. Venkataramani, "Disaster recovery as a cloud service: Economic benefits & deployment challenges," *Proc. of HotCloud, Boston, MA*, 2010.

[38] H. Schmeck, C. Müller-Schloer, E. Çakar, M. Mnif, and U. Richter, "Adaptivity and self-organization in organic computing systems," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 5, no. 3, p. 10, 2010.

[39] S. Dobson, R. Sterritt, P. Nixon, and M. Hinchey, "Fulfilling the vision of autonomic computing," *Computer*, vol. 43, no. 1, pp. 35–41, 2010.

[40] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[41] M. Caraman, S. Moraru, S. Dan, and C. Grama, "Continuous disaster tolerance in the iaas clouds," in *Optimization of Electrical and Electronic Equipment (OPTIM), 2012 13th International Conference on*. IEEE, 2012, pp. 1226–1232.

[42] U. F. Minhas, S. Rajagopalan, B. Cully, A. Aboulnaga, K. Salem, and A. Warfield, "Remusdb: Transparent high availability for database systems," in *Proc. of VLDB*, 2011.

[43] S. Rajagopalan, B. Cully, R. O'Connor, and A. Warfield, "Secondsite: disaster tolerance as a service," in *Proceedings of the 8th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments*. ACM, 2012, pp. 97–108.

[44] T. Wood, H. A. Lagar-Cavilla, K. Ramakrishnan, P. Shenoy, and J. Van der Merwe, "Pipecloud: using causality to overcome speed-of-light delays in cloud-based disaster recovery," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 2011, p. 17.

[45] A. ELB, "Amazon elastic load balancing," http://aws.amazon.com/elasticloadbalancing, April 2013.

[46] K. Begnum, N. A. Lartey, and L. Xing, "Cloud-oriented virtual machine management with mln," in *Cloud Computing*. Springer, 2009, pp. 266–277.

[47] M. Pokharel, S. Lee, and J. S. Park, "Disaster recovery for system architecture using cloud computing," in *Applications and the Internet (SAINT), 2010 10th IEEE/IPSJ International Symposium on*. IEEE, 2010, pp. 304–307.

[48] R. R. Schulman, "Disaster recovery issues and solutions," *Hitachi Data Systems White paper*, 2004.

[49] Y. Ueno, N. Miyaho, S. Suzuki, and K. Ichihara, "Performance evaluation of a disaster recovery system and practical network applications in cloud compting envionments," *International Journal On Advances in Networks and Services*, vol. 4, no. 1 and 2, pp. 130–137, 2011.

[50] J. J. Wylie, "Selecting the right data distribution scheme for a survivable storage system," 2001.

Chapter 9

Appendix

9.1 maintenance.pl

```
1  #!/usr/bin/perl
2
3  use strict;
4  use feature ":5.10";
5  use Getopt::Std;
6  $Getopt::Std::STANDARD_HELP_VERSION=1;
7  use Fcntl;
8  use NDBM_File;
9  use Sys::Hostname;
10 use feature "switch";
11
12 BEGIN:
13
14 $ENV{"EC2_PRIVATE_KEY"} = "/root/.amazon/pk.pem";
15 $ENV{"EC2_CERT"} = "/root/.amazon/cert.pem";
16 $ENV{"USER"} = "root";
17 $ENV{"LANGUAGE"} = "unset";
18 $ENV{"LC_ALL"} = "en_GB.utf8";
19 $ENV{"LANG"} = "en_US.UTF-8";
20 my $myname = hostname();
21 $myname =~ /controller(\d{1,3})\.(.+)/ ;
22 my $myrank = $1;
23 my $hostnumber;
24 my %opts;
25 getopts('p:n:', \%opts)|| usage();
26 my $project = "globalhydra";
27 $project = $opts{"p"} if (exists $opts{"p"});
28 my $EC2_number = 3;
29 $EC2_number = $opts{"n"} if (exists $opts{"n"});
30 usage() if ($EC2_number !~ /^\d{1,3}$/);
31 open(REP,">>","maintenance_report");
```

```
32  my @ins = (); my @cq_array = ();
33  my %hosts = (); my $same_instance = 1;
34  my $access=O_RDWR+O_CREAT;
35  tie(%hosts, "NDBM_File", "./hosts", $access, 0600);
36  my %Region_ami=("ap-northeast-1" => "ec2 {\n    region ap-
        northeast-1\n    ami ami-5bbc325a\n}\n",
37          "ap-southeast-1" => "ec2 {\n    region ap-southeast-1\n
                ami ami-000a4452\n}\n",
38          "ap-southeast-2" => "ec2 {\n    region ap-southeast-2\n
                ami ami-310d9d0b\n}\n",
39          "eu-west-1" => "ec2 {\n    region eu-west-1\n    ami ami-
                f97a6b8d\n}\n",
40          "sa-east-1" => "ec2 {\n    region sa-east-1\n    ami ami
                -8870aa95\n}\n",
41          "us-east-1" => "ec2 {\n    region us-east-1\n    ami ami-8
                df59be4\n}\n",
42          "us-west-1" => "ec2 {\n    region us-west-1\n    ami ami-2
                fb8976a\n}\n",
43          "us-west-2" => "ec2 {\n    region us-west-2\n    ami ami-
                a135a391\n}\n");
44
45  my %regions=("eu-west-1"=>0,"sa-east-1"=>0,"us-east-1"=>0,"ap-
        northeast-1"=>0,"us-west-2"=>0,"us-west-1"=>0,"ap-southeast-1"
        =>0,"ap-southeast-2"=>0);
46
47  Update_Hosts();
48  Instance_Information();
49  foreach my $key (keys %hosts){say REP "controller$key  $hosts{$key
        }{instance} ($hosts{$key}{pubIP})  ";}
50  Maintenance() if(Check_rank());
51  my $length = scalar(@cq_array);
52  say REP "cq_array length : $length\nhosts need to be create :
        @cq_array";
53  Creator(@cq_array) if (scalar(@cq_array)>0);
54  sub Record_Hosts    # need @hosts
55    {
56        my $IParray = $_[0];
57        open(HOSTS,"/etc/hosts");
58        while(<HOSTS>)
59    {
60        next unless /controller(\d{1,3})/;
61        next if /127/;
62        $$IParray[$1] = (split)[0];
63    }
64        close HOSTS;
65    }
66  sub Instance_Information
67    {
68
```

```
69          my @hosts; my $Pname;
70          Record_Hosts(\@hosts);
71          foreach my $reg (keys %regions)
72      {
73          open(DIN,"/usr/bin/ec2din --region $reg --filter \"tag-value
                =$project*\" |");
74          while (<DIN>)
75          {
76        next unless /^INSTANCE/; next if /terminated/;
77        $ins[1] = (split)[1];    ## instance's name
78        $ins[2] = 1 if /running/;    ##  check status
79        /201(\d)-(\d\d)-(\d\d)T(\d\d):(\d\d):(\d\d)/;
80        $ins[3] = $1.$2.$3.$4.$5.$6;          ## launch time
81        /((eu|sa|us|ap)-\w+-\d)\w*\s/;
82        $ins[4] = $1;             ## region
83        /(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\s+(\d{1,3}\.\d{1,3}\.\d
              {1,3}\.\d{1,3})/;
84        $ins[5] = $1;      ## pubIP
85        $ins[6] = $2;      ## priIP
86        while (<DIN>)
87        {
88            last if /^RESERVATION/;
89            next  unless /^TAG/;
90            $ins[7] = (split)[4];
91        }
92        if($ins[7] =~ /$project.controller(\d{1,3})/)
93        {
94            $Pname = "$project.m$1";
95            $ins[0] = $1;    ## host rank
96            if(defined($hosts{$ins[0]}))
97            {$hosts{$ins[0]}{number}++; $ins[0] .= "_".$hosts{$ins
                  [0]}{number};}
98            $hosts{$ins[0]}{instance} = $ins[1];
99            $hosts{$ins[0]}{status} = $ins[2];
100           $hosts{$ins[0]}{launch}  = $ins[3];
101           $hosts{$ins[0]}{region}  = $ins[4];
102           $hosts{$ins[0]}{pubIP}  = $ins[5]; $hostnumber = $ins[0]
                  if ($ins[5] eq $hosts[$myrank]);
103           $hosts{$ins[0]}{priIP}  = $ins[6];
104           $hosts{$ins[0]}{number} = $same_instance;
105           $hosts{$ins[0]}{Pname} = $Pname;
106       }
107       @ins = ();
108         }
109         close DIN;
110     }
111     }
112 sub Check_rank
113 {
```

```
114       for(my $n=1; $n<$myrank; $n++)
115       {
116     if($hosts{$n}{instance})
117     {
118         return 0 if $hosts{$n}{status};
119
120         foreach(2..$hosts{$n}{number})
121         {
122       my $m = $n."_$_";
123       return 0 if($hosts{$m}{status});
124         }
125     }
126       }
127       if($hosts{$myrank}{number}>1)   ## there are more than one of
              me
128       {
129     return 0 if ($hosts{$hostnumber}{launch} < $hosts{$myrank}{
            launch});
130     foreach(2..$hosts{$myrank}{number})     ## comparing launch time
131       {
132           my $m = $myrank."_$_";
133           return 0 if $hosts{$m}{launch}>$hosts{$hostnumber}{launch
                  };
134       }
135       }
136       return 1;
137   }
138   sub Maintenance
139     {
140         my @hosts; my $exist = 0;
141         Record_Hosts(\@hosts);
142         say REP "\n/etc/hosts:@hosts\n";     ## report
143         foreach my $m (1..$EC2_number)
144         {
145       say REP "This is \$hosts{$m}{instance}:$hosts{$m}{instance}";
146       unless($hosts{$m}{instance})   ## the controller do not exist
147       {
148           say REP "The one do not exits loop";
149           Terminate_All_Agents($m);
150           Creator_queue($m,\@cq_array);        ## put into create-
                  queue
151       }
152       elsif($hosts{$m}{number}>1)  ## exists but there are more than
              one have same name
153       {
154           say REP "The one more than one loop";
155           unless ($hosts{$m}{status}&&($hosts{$m}{pubIP} eq $hosts[
                  $m]))
156           {
```

```
157         Terminate($m);
158           }else {$exist = 1;}
159           foreach (2..$hosts{$m}{number})
160       {
161           my $n = $m."_".$_;
162           unless ($hosts{$n}{status}&&($hosts{$n}{pubIP} eq $hosts[
                  $m]))
163           {
164         Terminate($n);
165           }else{$exist = 1;}
166       }
167           Creator_queue($m,\@cq_array) unless $exist;  ## if not
                  exist set into create-queue
168       }
169       else
170       {
171           unless ($hosts{$m}{status}&&($hosts{$m}{pubIP} eq $hosts[
                  $m]))
172       {
173           say REP "The one not match hosts loop";
174           Terminate($m);
175           Creator_queue($m,\@cq_array);
176       }
177       }
178         }
179     }
180   sub Creator_queue     ##  argument $hostname, \@cq_array (edit
        original array)
181     {
182         my ($host,$cqarray) = @_;
183         $$cqarray[scalar(@$cqarray)] = $host;
184         say "The controller$host will be created.....";
185     }
186   sub Regions  ## argument \%regions
187   {
188       open(REG,"/usr/bin/ec2-describe-regions|");
189       my ($regions) = @_;
190       while (<REG>)
191       {
192     chomp($_);
193     $$regions{(split)[1]}=0;  ## get all available region names
194       }
195   }
196   sub Used_Regions    ##  need \%regions from Regions()
197     {
198         my ($regions) = @_;
199
200         foreach my $reg (keys %hosts)
201         {
```

```
202       $hosts{$reg}{region} =~ /((eu|sa|us|ap)-\w+-\d)\w*/;
203       $$regions{$1} += 1;
204       say REP "regionarray $1:$$regions{$1}";
205         }
206     }
207   sub Creator         ## argument @cq_array
208     {
209         my(@creator_queue,@sort_region);
210         my $n=0;
211         Regions(\%regions);           ## get all region names
212         Used_Regions(\%regions);      ## mark the rate of region use
213         @creator_queue = @_;          ## the hosts need to be created
214         open(MLNE,"/mnt/globalhydra.mln");
215         while(<MLNE>)
216     {
217         next unless /(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\s+(.*?
                controller\d{1,3}\.$project)/;
218         say REP "globalhydra.mln: IP: $1    name: $2 myIP: $hosts{
                $hostnumber}{pubIP} myname: $myname";
219         system("/bin/sed -i s/$1/$hosts{$hostnumber}{pubIP}/g /mnt/
                globalhydra.mln");
220         system("/bin/sed -i s/$2/$myname/g /mnt/globalhydra.mln");
221     }
222         close MLNE;
223         foreach my $key (sort {$regions{$a} <=> $regions{$b}} keys %
                regions) ##  sort regions hash
224     {
225         $sort_region[$n]=$key;
226         $n++;
227     }
228
229         for(my $m=0,my $a=0; $a<scalar(@creator_queue); $m++,$a++)
230     {
231         Region_File_Change($creator_queue[$a],$sort_region[$m]); ##
                region file changed
232         given ($creator_queue[$a])
233           {
234         $myname =~ /(.*?controller)\d{1,3}\.$project/;
235         my $mname = $1;
236         when ("1")
237         {system("/usr/bin/mln upgrade -f /mnt/globalhydra.mln -l ${
                mname}2:${mname}3");system("/usr/bin/mln start -p
                $project -h ${mname}1");}
238         when ("2")
239         {system("/usr/bin/mln upgrade -f /mnt/globalhydra.mln -l ${
                mname}1:${mname}3");system("/usr/bin/mln start -p
                $project -h ${mname}2");}
240         when ("3")
241         {system("/usr/bin/mln upgrade -f /mnt/globalhydra.mln -l ${
```

```
            mname}1:${mname}2");system("/usr/bin/mln start -p
            $project -h ${mname}3");}
242           }
243         if($m+2>scalar(@sort_region)){$m=0;}     ##  if region array
              is end ,restart from begining
244     }
245         Glusterfs_Rebuild();
246     }
247   sub Glusterfs_Rebuild
248     {
249         open(HOSTG,"</etc/hosts");
250         my @hostsfile = <HOSTG>;
251         close HOSTG;
252         open(HOSTG,">/etc/hosts");
253         open(OPTHOSTS,">/root/hosts");
254         $myname =~ /^(.*?controller)\d{1,3}(.+)$/;
255         foreach(@cq_array)
256     {
257         system("/usr/bin/hydra cert clean $1.$_.$2");
258         say "Starting  gluster volume remove-brick hydra
              controller$_:/brick";
259         system("/usr/bin/expect -c \"spawn /usr/sbin/gluster volume
              remove-brick hydra controller$_:/brick; expect (y/n);
              send y\\r;interact\"");
260         system("/usr/sbin/gluster peer detach controller$_");
261     }
262         foreach my $LINE (@hostsfile)
263           {
264       print HOSTG $LINE unless (($LINE =~ /controller/)&&($LINE !~
            /127/));
265     }
266         my($pubIP,$hostname,$host,$n);
267         foreach my $reg (keys %regions)
268           {
269   REDO:    if ($n<6)  ## retry 5 times to make sure all hosts is up
270         {
271             system("/usr/bin/ec2din --region $reg --filter \"tag-
                  value=$project*\" > /root/Glusterfs_R_report");
272             open(NEW,"/root/Glusterfs_R_report");
273             while(<NEW>)
274             {
275           if (/stop/)
276           {
277               close NEW;
278               goto BEGIN;
279           }
280           if (/pending/)
281           {
282               close NEW;
```

```
283                 $n++;
284                 goto REDO;
285             }
286               }
287               close NEW;
288               goto START;
289           }
290   START:       open(NEW,"/root/Glusterfs_R_report");
291           while(<NEW>)
292           {
293               next unless /running/;
294               $pubIP = 0; $hostname = 0;
295               /(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\s+(\d{1,3}\.\d
                      {1,3}\.\d{1,3}\.\d{1,3})/;
296               $pubIP = $1;
297               while (<NEW>)
298               {
299             last if /^RESERVATION/;
300             next  unless /$project\..*?(controller\d{1,3})/;
301             $hostname = $1;
302               }
303               say HOSTG "$pubIP $hostname" if ($pubIP&&$hostname);
304               say OPTHOSTS "$pubIP $hostname" if ($pubIP&&$hostname);
305           }
306           close NEW;
307             }
308           close HOSTG; close OPTHOSTS;
309           system("/bin/cp /root/hosts /mnt/hosts");
310           say "Waiting for the New controllers ready...."; sleep 100;
311           foreach(@cq_array)
312           {
313         system("/usr/sbin/gluster peer probe controller$_");
314         sleep 10;
315         system("/usr/sbin/gluster volume add-brick hydra controller$_
              :/brick");
316           }
317           system("/usr/sbin/gluster volume rebalance hydra start");
318     }
319   sub Region_File_Change
320     {
321           open(REGION,">/opt/hydracontroller$_[0].mln");
322           say REGION $Region_ami{$_[1]};
323           close REGION;
324     }
325   sub Update_Hosts
326   {
327         my @opt; my $n = 0;
328         open(HOSTS,"/etc/hosts");
329         open(OPTHOSTS,"/mnt/hosts");
```

```
330       while(<HOSTS>)
331       {
332     next if (($_ =~ /controller/)&&($_ !~ /127/));
333     $opt[$n] = $_;
334           $n++;
335       }
336       close HOSTS;
337       while(<OPTHOSTS>)
338       {
339     next unless /controller(\d{1,3})/;
340     $opt[$n] = $_;
341     $n++;
342       }
343       close OPTHOSTS;
344       open(HOSTS,">/etc/hosts");
345       foreach (@opt)
346       {
347     print HOSTS $_;
348       }
349       close HOSTS;
350   }
351   sub Terminate_All_Agents
352     {
353         my $ins = 0;
354         foreach my $reg (keys %regions)
355         {
356       open(DIN,"/usr/bin/ec2din --region $reg --filter \"tag-value=
              $project.m$_[0]*\" |");
357
358       while (<DIN>)
359       {
360           next unless /^INSTANCE/; next if /terminated/;
361           $ins = (split)[1];
362           system("/usr/bin/ec2-terminate-instances $ins --region
                $reg");
363           say REP "This $ins needs to be kill";
364       }
365         }
366     }
367   sub Terminate
368   {
369       my(@terminate,$ins,$name,$n);
370       $n = 0;
371       $terminate[$n] = $hosts{$_[0]}{instance};
372       open(DIN,"/usr/bin/ec2din --region $hosts{$_[0]}{region} --
              filter \"tag-value=$project.m*\" |");  # find the project
              in controller's region first
373       while(<DIN>)
374       {
```

```
375             next unless /^INSTANCE/; next if /terminated/;
376             $ins = (split)[1];
377             while(<DIN>)
378             {
379                 last if /^RESERVATION/;
380                 next  unless /^TAG/;
381                 $name = (split)[4];
382                 if ($name =~ /$hosts{$_[0]}{Pname}/)
383           {
384         $n++;
385         $terminate[$n] = $ins;
386           }
387       }
388         }
389         foreach (@terminate)
390         {
391       system("/usr/bin/ec2-terminate-instances $_ --region $hosts{$_
              [0]}{region}");
392         }
393         close DIN;
394   }
395   END:
396   untie(%hosts);
397   close REP;
398   close REG;
399   sub usage
400       {
401                 say "Usage:";
402                 say "-p project -n number of host (1-999)";
403                 say "The name of project and the number of controllers
                        is required";
404                 exit ;
405        }
```

9.2 mastercontrolling

```
1   #!/usr/bin/perl
2   use strict;
3   use feature ":5.10";
4   use Sys::Hostname;
5   use feature "switch";
6   use Tie::File;
7   BEGIN:
8   $ENV{"EC2_PRIVATE_KEY"} = "/root/.amazon/pk.pem";
9   $ENV{"EC2_CERT"} = "/root/.amazon/cert.pem";
10  $ENV{"USER"} = "root";
11  $ENV{"LANGUAGE"} = "unset";
12  $ENV{"LC_ALL"} = "en_GB.utf8";
13  $ENV{"LANG"} = "en_US.UTF-8";
```

```
14  my @ins;
15  my @mysite;
16  my $myname = hostname();
17  $myname =~ /controller(\d{1,3})\.(.+)/ ;
18  my $rank = $1;
19  my $project = $2;
20  open(REP,">>","mastercontrolling_report");
21  my %Region_ami=("ap-northeast-1" => "ec2 {\n    region ap-
        northeast-1\n    ami ami-5bbc325a\n}\n",
22                  "ap-southeast-1" => "ec2 {\n    region ap-southeast-1\
                        n    ami ami-000a4452\n}\n",
23                  "ap-southeast-2" => "ec2 {\n    region ap-southeast-2\
                        n    ami ami-310d9d0b\n}\n",
24                  "eu-west-1" => "ec2 {\n    region eu-west-1\n    ami
                        ami-f97a6b8d\n}\n",
25                  "sa-east-1" => "ec2 {\n    region sa-east-1\n    ami
                        ami-8870aa95\n}\n",
26                  "us-east-1" => "ec2 {\n    region us-east-1\n    ami
                        ami-8df59be4\n}\n",
27                  "us-west-1" => "ec2 {\n    region us-west-1\n    ami
                        ami-2fb8976a\n}\n",
28                  "us-west-2" => "ec2 {\n    region us-west-2\n    ami
                        ami-a135a391\n}\n");
29  my @webservern; my $webserverdb='./webserver.db';
30  tie(@webservern,'Tie::File',$webserverdb) or die;
31  my ($reg,$IP) = My_Region();
32  say "I am in region : $reg\nMy IP is : $IP";
33  if (Test_Controller())
34    {Test_Webservers();Rebuild();}
35  else
36    {Terminate_All_Agents();Rebuild();}
37  sub My_Region
38    {
39        my $region; my $priIP;
40        open(META,"/usr/bin/ec2metadata |");
41        while(<META>)
42        {
43      $region = $1 if  /((eu|sa|us|ap)-\w+-\d)\w*\s/;
44      $priIP = (split)[1]  if /local-ipv4/;
45        }
46        close META;
47        return $region,$priIP;
48    }
49  sub Test_Controller
50    {
51        my($n,$sign);
52        REDO: if ($n < 6)
53        {
54      $sign = 0;
```

```
55      system("/usr/bin/ec2din --region $reg --filter \"tag-value=
            $project\.m$rank*\" > /root/agentstemp"); # globalhydra.
            controller1.something
56      open(DIN,"/root/agentstemp");
57      while (<DIN>)
58      {
59          @ins = ();
60          next unless /^INSTANCE/; next if /terminated/;
61          if (/pending/)
62          {
63        close DIN;
64        $n++;
65        goto REDO;
66          }
67          $ins[1] = (split)[1];   ## instance's name
68          $ins[2] = 1 if /running/;
69
70          while(<DIN>)
71          {
72        last if /^RESERVATION/;
73        next  unless /^TAG/;
74        $ins[3] = (split)[4];
75       $sign = 1 if ((/$project\.m$rank\.master/)&&$ins[2])
76          }
77      }
78        }else {exit;}
79        close DIN;
80        return $sign;
81  }
82  sub Test_Webservers
83  {
84      open(DIN,"/usr/bin/ec2din --region $reg --filter \"tag-value=
            $project\.m$rank*\" |");
85      @ins =(); @mysite = ();
86      while(<DIN>)
87      {
88    next unless /^INSTANCE/; next if /terminated/;
89    $ins[1] = (split)[1];   ## instance's name
90    if (/running/)
91    {
92          $ins[2] = 1;                  ## instance's status
93          /(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\s+(\d{1,3}\.\d{1,3}\.\
              d{1,3}\.\d{1,3})/;
94          $ins[5] = $1;      ## pubIP
95          $ins[6] = $2;      ## priIP
96          while (<DIN>)
97          {
98      last if /^RESERVATION/;
99      next  unless /^TAG/;
```

```
100     $ins[3] = (split)[4];
101     given ($ins[3])
102     {
103         when (/master/)
104           {
105         $mysite[0][0] = 1;   ## the master is alive
106         if ($mysite[0][1])    ## there is one master already
                registed
107         {
108             @mysite = ();
109             Terminate_All_Agents();
110             Rebuild();
111         exit;
112         }
113         $mysite[0][3] = $ins[6];
114           }
115         when (/webserverA/)
116         {
117
118       $mysite[1][0] = 1;   ## the webserverA is alive
119       if ($mysite[1][1])    ## there is one webserverA already
              registed
120       {
121           $mysite[1][1] =~ /webserverA(\d{1,3})/;
122           if ($mysite[1][1] < $1)
123             {
124           Terminate($mysite[1][2]);
125           $mysite[1][1] = $1;
126           $mysite[1][2] = $ins[1];
127             }else{ Terminate($ins[1]);}
128       }
129           }
130           when (/webserverB/)
131           {
132
133       $mysite[2][0] = 1;   ## the webserverB is alive
134       if ($mysite[2][1])    ## there is one webserverB already
              registed
135           {
136       $mysite[2][1] =~ /webserverB(\d{1,3})/;
137                           if ($mysite[2][1] < $1)
138       {
139           Terminate($mysite[2][2]);
140           $mysite[2][1] = $1;
141           $mysite[2][2] = $ins[1];
142       }else{ Terminate($ins[1]);}
143           }
144       }
145           }
```

```
146         }
147           }else{Terminate($ins[1]);}
148           }
149       }
150   sub Terminate_All_Agents
151   {
152       open(DIN,"/usr/bin/ec2din --region $reg --filter \"tag-value=
              $project\.m$rank*\" |");
153       while (<DIN>)
154       {
155     @ins = ();
156     next unless /^INSTANCE/; next if /terminated/;
157     $ins[1] = (split)[1];
158     while(<DIN>)
159     {
160         last if /^RESERVATION/;
161         next  unless /^TAG/;
162         $ins[2] = (split)[4];
163         Terminate($ins[1]) if ((/master/)||/webserver/);
164     }
165       }
166       close DIN;
167       $webservern[1]=0;
168       $webservern[2]=0;
169   }
170   sub Terminate
171   {
172       system("/usr/bin/ec2-terminate-instances $_[0] --region $reg")
              ;
173   }
174   sub Rebuild
175   {
176       my $masterIP = $mysite[0][3];
177       unless ($mysite[0][0])
178       {
179     system("/usr/bin/hydra cert clean master.$project.m$rank");
180     system("/bin/sed -i \"s/controller[0-9]\\{1,3\\}.$project/
              controller$rank.$project/\" /root/master.mln");
181     system("/bin/sed -i \"s/$project.m[0-9]\\{1,3\\}/$project.m$rank
              /\" /root/master.mln");
182     system("/bin/sed -i \"s
              /[0-9]\\{1,3\\}.[0-9]\\{1,3\\}.[0-9]\\{1,3\\}.[0-9]\\{1,3\\}/
              $IP/g\" /root/master.mln");
183     open(REGION,">/root/master_region.mln");
184     say REGION $Region_ami{$reg};
185     close REGION;
186     system("/usr/bin/expect -c \"spawn /usr/bin/mln build -f /root/
              master.mln; expect (y/n); send y\\r;interact\"");
187     system("/usr/bin/mln start -p $project\.m$rank");
```

```
188     sleep 40;
189     for(my $n=0,my $m=0;$n==0&&$m<6;$m++)
190       {
191           open(DIN,"/usr/bin/ec2din --region $reg --filter \"tag-
                  value=$project\.m$rank\.master\" |");
192           while (<DIN>)
193           {
194         /(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\s+(\d{1,3}\.\d{1,3}\.\
                 d{1,3}\.\d{1,3})/;
195         $masterIP = $2;
196         $n++ if $masterIP;
197           }
198           close DIN;
199       }
200         }
201       unless ($mysite[1][0])
202         {
203       $webservern[1]++;
204       system("/bin/sed -i \"s/$project.m[0-9]\\{1,3\\}/$project.
              m$rank/\" /root/webserverA.mln");
205       system("/bin/sed -i \"s/master.$project.m[0-9]\\{1,3\\}/master
              .$project.m$rank/\" /root/webserverA.mln"); ## master.
              globalhydra.ml
206       system("/bin/sed -i \"s
              /[0-9]\\{1,3\\}.[0-9]\\{1,3\\}.[0-9]\\{1,3\\}.[0-9]\\{1,3\\}/
              $masterIP/g\" /root/webserverA.mln");
207       system("/bin/sed -i \"s/webserverA[0-9]\\{1,3\\}/
              webserverA$webservern[1]/\" /root/webserverA.mln");
208       system("/bin/rm -r /opt/mln/projects/root/$project.m${rank}A")
              ;
209       system("/usr/bin/mln build -f /root/webserverA.mln");
210       system("/usr/bin/mln start -p $project\.m${rank}A");
211         }
212       unless ($mysite[2][0])
213       {
214     $webservern[2]++;
215     system("/bin/sed -i \"s/$project.m[0-9]\\{1,3\\}/$project.m$rank
          /\" /root/webserverB.mln");
216     system("/bin/sed -i \"s/master.$project.m[0-9]\\{1,3\\}/master.
          $project.m$rank/\" /root/webserverB.mln"); ## master.
          globalhydra.ml
217     system("/bin/sed -i \"s
          /[0-9]\\{1,3\\}.[0-9]\\{1,3\\}.[0-9]\\{1,3\\}.[0-9]\\{1,3\\}/
          $masterIP/g\" /root/webserverB.mln");
218     system("/bin/sed -i \"s/webserverB[0-9]\\{1,3\\}/
          webserverB$webservern[2]/\" /root/webserverB.mln");
219     system("/bin/rm -r /opt/mln/projects/root/$project.m${rank}B");
220     system("/usr/bin/mln build -f /root/webserverB.mln");
221     system("/usr/bin/mln start -p $project\.m${rank}B");
```

```
222         }
223     }
224     END:
225     close REP;
226     untie(@webservern);
```

9.3 initialization.pl

```
1   #!/usr/bin/perl
2   use strict;
3   use feature ":5.10";
4   use Sys::Hostname;
5   $ENV{"EC2_PRIVATE_KEY"} = "/root/.amazon/pk.pem";
6   $ENV{"EC2_CERT"} = "/root/.amazon/cert.pem";
7   open(LOG,">/root/pre-ready.log");
8   open(HOSTS,">>","/etc/hosts") || say LOG "could not open /etc/
        hosts";
9   open(OPTHOSTS,">","/root/hosts");
10  my $myname = hostname();
11  $myname =~ /.*(controller\d{1,3})\.(.+)/ ;
12  $myname = $1;
13  say HOSTS "127.0.0.1 $myname";
14  my $project = $2;
15  my($pubIP,$Name);
16  my @region = qw(eu-west-1 sa-east-1 us-east-1 ap-northeast-1 us-
        west-2 us-west-1 ap-southeast-1 ap-southeast-2);
17  say LOG @region;
18  foreach my $reg (@region)
19    {
20        open(DIN,"/usr/bin/ec2din --region $reg --filter \"tag-value
              =$project*\" |") || say LOG "ec2din command failed"; ##
              amazon has different order with name
21        while (<DIN>)
22        {
23      next unless /^INSTANCE/; next if /terminated/; next if /
            stopped/;
24      say LOG "find the instance";
25      /(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\s(\d{1,3}\.\d{1,3}\.\d
            {1,3}\.\d{1,3})/;
26      $pubIP = $1; next unless $pubIP; say LOG "The public IP is
            $pubIP";
27      while (<DIN>)
28      {
29          last if /^RESERVATION/;
30          next unless /^TAG/;     say LOG "Get into the second DIN
                loop";
31          /$project\..*?(controller\d{1,3})/;
32          $Name = $1; say LOG "The hostname is $Name";
33          say HOSTS "$pubIP $Name" if $Name;
```

```
34            say OPTHOSTS "$pubIP $Name" if $Name;
35        }
36          }
37    }
38    close LOG;
39    close(DIN);
40    close HOSTS;
41    close OPTHOSTS;
```

9.4 Puppet Manifests

9.4.1 Hydra Master Initialization

```
1   class hydracontroller::install {
2       File {
3           require => Class["mln"],
4       }
5       file { "/etc/default/puppetmaster" :
6         owner => "root",
7         group => "root",
8         mode => 0640,
9         source => "puppet:///modules/hydracontroller/puppetmaster",
10        }
11      file { "/usr/bin/hydra" :
12        owner => "root",
13        group => "root",
14        mode => 0755,
15        source => "puppet:///modules/hydracontroller/hydra",
16        }
17     file { "/root/.bashrc" :
18        owner => "root",
19        group => "root",
20        mode => 0755,
21        source => "puppet:///modules/hydracontroller/.bashrc",
22        }
23        file { "/etc/hydra" :
24        ensure => directory,
25        recurse => true,
26        owner => "root",
27        group => "root",
28        mode => 0755,
29        source => "puppet:///modules/hydracontroller/puppet",
30        }
31       file { "/brick" :
32        ensure => directory,
33        recurse => true,
34        owner => "root",
35        group => "root",
36        mode => 0755,
```

```
37          }
38        file { "/etc/hydra/puppet.conf" :
39          owner => "root",
40          group => "root",
41          mode => 0640,
42          content => template("hydracontroller/puppet.conf.erb"),
43          require => File["/etc/hydra"],
44        }
45       package { "puppetmaster" :
46          ensure => present,
47          require => File["/etc/default/puppetmaster","/etc/hydra/
                puppet.conf"],
48       }
49      package { "expect" :
50            ensure => present,
51         }
52      package { "glusterfs-server" :
53          ensure => present,
54       }
55      package { "glusterfs-client" :
56          ensure => present,
57       }
58        file { "/root/pre-ready.pl" :
59              owner => "root",
60        group => "root",
61        mode => 0755,
62        source => "puppet:///modules/hydracontroller/script/pre-ready.
              pl",
63        }
64    }
```

9.4.2 Glusterfs Initialization

```
1   ## the gluster_init will generate a report as /root/
        gluster_finished
2   # /brick have been created in hydracontroller::install
3   class hydracontroller::preinit {
4            exec { "/root/pre-ready.pl":
5             unless => "/usr/bin/test -e /root/pre-ready.log",
6             path => "/usr/bin/:/usr/sbin/:/usr/local/bin:/bin:/sbin"
                  ,
7             require => Class['hydracontroller::install','mln::
                  install'],
8             }
9   }
10  class hydracontroller::glusterfs {
11       case $hostname {
12             /.*controller1.*/:{
13                  file { "/root/gluster_init" :
```

```
14            owner => "root",
15            group => "root",
16            mode => 0755,
17            source => "puppet:///modules/hydracontroller/script/
                  gluster_init",
18            require => Class["hydracontroller::preinit"],
19            }
20                    exec { "/root/gluster_init":
21       unless => "/usr/bin/test -e /root/gluster_init.log",
22       require => File["/root/gluster_init"],
23       }
24        }
25              /.*controller[02-9].*/:{
26            file { "/root/gluster_sleep" :
27            owner => "root",
28            group => "root",
29            mode => 0755,
30            source => "puppet:///modules/hydracontroller/script/
                  gluster_sleep",
31            require => Class["hydracontroller::preinit"],
32            }
33            exec { "/root/gluster_sleep":
34            unless => "/usr/bin/test -e /root/gluster_sleep.log",
35            require => File["/root/gluster_sleep"],
36            }
37         }
38     }
39    }
40   ## for all hosts
41       class  hydracontroller::glustermount {
42         file { "/root/mount.pl" :
43         owner => "root",
44         group => "root",
45         mode => 0755,
46         source => "puppet:///modules/hydracontroller/script/mount.pl
                  ",
47         require => Class["hydracontroller::glusterfs"],
48         }
49       cron { mount :
50              command => "/root/mount.pl",
51        user    => root,
52        minute  => '*/6'
53        }
54          exec { "/root/mount.pl":
55          require => File["/root/mount.pl"],
56     }
57   }
58   ##  make  glusterfs /mnt  only for node controller1
59   class hydracontroller::glusterfscp{
```

```
60          file { "/opt/mln":
61           ensure => "directory",
62           owner  => "root",
63           group => "root",
64           mode => 0640,
65           require => Class["hydracontroller::glustermount"],
66           }
67         file { "/opt/mln/templates" :
68         ensure => "directory",
69         owner => "root",
70         group => "root",
71         mode => 0640,
72         require => File["/opt/mln"],
73         }
74         file { "/opt/mln/projects" :
75         ensure => "directory",
76         owner => "root",
77         group => "root",
78         mode => 0640,
79         require => File["/opt/mln"],
80         }
81         file { "/opt/mln/files" :
82         ensure => "directory",
83         owner => "root",
84         group => "root",
85         mode => 0640,
86         require => File["/opt/mln"],
87         }
88  }
```

9.4.3 Maintenance Preparation

```
1   class maintenance::install {
2       File {
3               require => Class['hydracontroller::glustermount','mln::
                    install'],
4        }
5   ###############    mln  ########
6   ## check if command running more than once
7           file { "/mnt/globalhydra.mln" :
8             owner => "root",
9                   group => "root",
10                  mode => 0755,
11                  source => "puppet:///modules/maintenance/
                        globalhydra.mln",
12                  require => Class["hydracontroller::glustermount"],
13                  }
14          file { "/opt/hydracontroller1.mln" :
15                  owner => "root",
```

```
16            group => "root",
17            mode => 0755,
18            source => "puppet:///modules/maintenance/hydracontroller.
                  mln",
19                  require => Class["hydracontroller::glustermount"],
20            }
21          file { "/opt/hydracontroller2.mln" :
22                  owner => "root",
23                  group => "root",
24                  mode => 0755,
25                  source => "puppet:///modules/maintenance/
                        hydracontroller.mln",
26                  require => Class["hydracontroller::glustermount"],
27                  }
28          file { "/opt/hydracontroller3.mln" :
29                  owner => "root",
30                  group => "root",
31                  mode => 0755,
32                  source => "puppet:///modules/maintenance/
                        hydracontroller.mln",
33                  require => Class["hydracontroller::glustermount"],
34                  }
35          file { "/opt/mln_build" :
36                  owner => "root",
37                  group => "root",
38                  mode => 0755,
39          source => "puppet:///modules/maintenance/mln_build",
40                  require => Class["hydracontroller::glustermount"],
41                  }
42                exec {"/opt/mln_build" :
43                unless => "/usr/bin/test -d /opt/mln/projects/root",
44                require => File[ "/opt/mln_build","/mnt/globalhydra.
                      mln","/opt/hydracontroller3.mln","/opt/
                      hydracontroller2.mln","/opt/hydracontroller1.mln"
                      ],
45          }
46          file { "/mnt/maintenance.pl" :
47                owner => "root",
48         group => "root",
49         mode => 0755,
50         source => "puppet:///modules/maintenance/maintenance.pl",
51         require => Class["hydracontroller::glustermount"],
52         }
53        cron { maintenance :
54              command => "/mnt/maintenance.pl",
55          user    => root,
56          minute  => '*/18',
57          require => File["/mnt/maintenance.pl"],
58          }
```

```
59      file { "/root/mastercontrolling.pl" :
60              owner => "root",
61        group => "root",
62        mode => 0755,
63        source =>"puppet:///modules/maintenance/mastercontrolling.pl
              ",
64              }
65       cron { mastercontrolling :
66              command => "/root/mastercontrolling.pl",
67        user    => root,
68        minute  => '*/18',
69        require => File["/root/mastercontrolling.pl"],
70        }
71      file { "/root/master.mln" :
72            owner => "root",
73            group => "root",
74            mode => 0755,
75            source => "puppet:///modules/maintenance/master.mln",
76            }
77       file { "/root/webserverA.mln" :
78       owner => "root",
79       group => "root",
80       mode => 0755,
81       source => "puppet:///modules/maintenance/webserverA.mln",
82            }
83       file { "/root/webserverB.mln" :
84       owner => "root",
85       group => "root",
86       mode => 0755,
87       source => "puppet:///modules/maintenance/webserverB.mln",
88       }
89       file { "/root/index.pl" :
90            owner => "root",
91      group => "root",
92      mode => 0755,
93      source => "puppet:///modules/maintenance/index.pl",
94      }
95   }
```

9.4.4 MLN Initialization

```
1   class mln::install {
2            file { "/root/bootstrap_hydra.sh" :
3            owner => "root",
4       group => "root",
5       mode => 0750,
6       source => "puppet:///modules/mln/bootstrap_hydra.sh",
7       }
8            exec { "/root/bootstrap_hydra.sh":
```

```
9             unless => "/usr/bin/test -e /root/boot_hydra_finished",
10          path => "/usr/bin/:/usr/sbin/:/usr/local/bin:/bin:/sbin",
11          require => File["/root/bootstrap_hydra.sh"],
12            }
13        package { "language-pack-en":
14            ensure => present,
15           }
16        package { "jed":
17        ensure => present,
18        }
19        package { "kpartx":
20        ensure => present,
21        }
22        package { "ec2-api-tools":
23        ensure => present,
24        require => Exec["/root/bootstrap_hydra.sh"],
25        }
26        package { "ec2-ami-tools":
27        ensure => present,
28        require => Exec["/root/bootstrap_hydra.sh"],
29        }
30        file { "/usr/bin/mln" :
31         owner => "root",
32         group => "root",
33         mode => 0750,
34         source => "puppet:///modules/mln/mln",
35         }
36        file { "/root/example_ec2.mln" :
37         owner => "root",
38         group => "root",
39         mode => 0644,
40         source => "puppet:///modules/mln/example_ec2.mln",
41         }
42         file { "/etc/mln/plugins" :
43         ensure => directory,
44         recurse => true,
45         owner => "root",
46         group => "root",
47         mode => 0640,
48         source => "puppet:///modules/mln/plugins",
49         require => File["/etc/mln"],
50         }
51        file { "/root/.amazon" :
52         ensure => directory,
53         recurse => true,
54         owner => "root",
55         group => "root",
56         mode => 0640,
57         source => "puppet:///modules/mln/.amazon",
```

```
58          require => File["/etc/mln"],
59          }
60          file { "/etc/mln" :
61          ensure => "directory",
62          owner => "root",
63          group => "root",
64          mode => 0640,
65          }
66          file { "/etc/mln/mln.conf" :
67          owner => "root",
68          group => "root",
69          mode => 0640,
70          content => template("mln/mln.conf.erb"),
71          require => File["/etc/mln"],
72          }
73
74          file { "/etc/libvirt/qemu.conf" :
75          owner => "root",
76          group => "root",
77          mode => 0640,
78          notify => [ Service["sanlock"], Service["libvirtd"] ],
79          content => template("mln/qemu.conf.erb"),
80          require => [ File["/etc/mln"], Mount["/opt/mln/sanlock"] ],
81        }
82          file { "/etc/libvirt/qemu-sanlock.conf" :
83          owner => "root",
84          group => "root",
85          mode => 0640,
86          notify => [ Service["sanlock"], Service["libvirtd"] ],
87          content => template("mln/qemu-sanlock.conf.erb"),
88          require => [ File["/etc/mln"], Mount["/opt/mln/sanlock"] ],
89          }
90  }
```

9.5 Example of MLN Files

9.5.1 globalhydra.mln

```
1   global {
2       project globalhydra
3   }
4   superclass common {
5      ec2 {
6          type ml.small
7          user_file {
8          echo $hostname.$project > /etc/hostname
9          hostname $hostname.$project
10         apt-get update
11         apt-get -y install puppet
```

```
12          echo "127.0.0.1 $hostname" >> /etc/hosts
13          echo "84.215.199.143   workstation" >> /etc/hosts
14          puppet agent --verbose --no-daemonize -o -w 60 --server=
                workstation
15          }
16          key ke-hydra
17      }
18      }
19  host controller1 {
20       superclass common
21  #include /root/hydramaster1.mln
22  }
23  host controller2 {
24       superclass common
25  #include /root/hydramaster2.mln
26  }
27  host controller3 {
28      superclass common
29  #include /root/hydramaster3.mln
30  }
```